UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Kravcar

Sinhronizacija heterogenih podatkovnih sistemov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Kravcar

Sinhronizacija heterogenih podatkovnih sistemov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Matjaž Kukar

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika diplomskega dela:

Sinhronizacija heterogenih podatkovnih sistemov je eden izmed pomembnih elementov v sodobnih, pogosto porazdeljeno implementiranih, informacijskih sistemov, obenem pa ima ključno vlogo pri njihovem povezovanju. Kandidat naj definira in podrobno preuči področje sinhronizacije heterogenih podatkovnih sistemov, opiše različne paradigme in na konkretnem primeru izvede implementacijo ter vrednotenje implementirane rešitve.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Tomaž Kravcar sem avtor diplomskega dela z naslovom:

Sinhronizacija heterogenih podatkovnih sistemov.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu prek univerzitetnega spletnega arhiva.

V Ljubljani, dne 9. oktobra 2015 Podpis avtorja:

Kazalo

Seznam slik

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADD-ON	Addition	Dodatki
API	Application programming interface	Programski vmesnik aplikacije
B2B	Business to business	Prodaja tipa podjetje–podjetje
B2C	Business to customer	Prodaja tipa podjetje–končna stranka
CSV	Comma separated values	Vrednosti, ločene z vejico
FTP	File Transfer Protocol	Protokol za izmenjavo datotek
GNU	GNU General Public License	GNU General Public License
IDE	Integrated development environment	Integrirano razvojno okolje
JDBC	Java database connectivity technology	Standard Java za izmenjavo podatkov
LAMP	Linux, Apache, MySQL, Perl/PHP/Python	Linux, Apache, MySQL, Perl/PHP/Python
ODBC	Open Database Connectivity	Odprti standard za izmenjavo podatkov
OS	Operating system	Operacijski sistem
PIS	Business operating system	Poslovno-informacijski sistem
RDBMS	Relational database management system	Sistem za upravljanje relacijskih baz podatkov
REST	Representational state transfer	Reprezentacija stanja prenosa
SAAS	Software as a service	Programska oprema kot storitev
SME	Small to medium sized enterprises	Majhna in srednje velika podjetja
SQL	Structured Query Language	Strukturno-poizvedbeni jezik
SSMS	SQL Server Management Studio	Urejevalnik za Microsoftov strežnik

		SQL
ST	Web store	Spletne trgovine
VCL	Visual component library	Vizualna knjižnica komponent
XLS	Microsoft Excel spreadsheet	Datoteka programa Microsoft Excel
XML	Extensible Markup Language	Razširjen označevalni jezik

Povzetek

Diplomsko delo opisuje mehanizem sinhronizacije podatkov med različnimi relacijskimi podatkovnimi strežniki na strani poslovno-informacijskega sistema (PIS) in različnimi relacijskimi podatkovnimi strežniki na strani spletne trgovine (ST). To z drugimi besedami pomeni, da opisujemo sinhronizacijo katerega koli PIS s katero koli ST. Enak oz. zelo podoben koncept bi se lahko uporabljal tudi v primeru sinhronizacije drugih informacijskih sistemov.

Sinhronizacija PIS in ST mora biti seveda zastavljena mnogo širše, kot je samo izmenjava podatkov na nivoju relacijske baze, vendar diplomsko delo opisuje predvsem ta nivo z nekaterimi podpornimi mehanizmi v okolju Delphi. Na hitro se je dotaknilo tudi ostalih mehanizmov, ki so potrebni za delovanje sinhronizacije kot celote.

Vsebinsko se je najprej predstavil globalni koncept delovanja, ki se ga v nadaljevanju bolj podrobno obrazloži, na koncu pa se predstavi tudi na konkretnem primeru uporabe. Funkcionalno se bomo omejili na sinhronizacijo zaloge, cen in naročil, ki predstavljajo osnovo sinhronizacije med PIS in ST, vendar lahko enako oz. podobno logiko apliciramo tudi na ostale funkcionalnosti oz. področja sinhronizacije.

Osnovna ideja o načinu sinhronizacije je nastala na podlagi praktičnih izkušenj in dejanskih potreb pri različnih poslovnih subjektih, ki so uporabljali programski paket Pantheon, vendar je z vidika uporabnosti za namene diplomskega dela dodatno razširjena in upamo, da bo v polni moči zaživela enkrat v prihodnosti.

Ključne besede: spletna trgovina, poslovno-informacijski sistem, sinhronizacija.

Abstract

The thesis describes the mechanism for data synchronisation between different relational data servers of a business operating system on one side and different relational data servers of a web store on the other. In other words, this denotes the synchronisation between any business operating system with any web store. In addition, a very similar concept could be introduced for synchronisation of other information systems.

However, the synchronisation between a business operating server and a web store must have a wider scope than the mere data exchange at the level of the relational database. Nevertheless, the thesis will mainly deal with this level and certain support mechanisms in the Delphi environment, and briefly mention other mechanisms necessary for the synchronisation as a whole.

In terms of content, the thesis first presents the global functioning concept, which is explained in more detail later on and used as a concrete example. As for the functional aspect, the thesis focuses on the synchronisation of stock, prices and orders, which are the core of the synchronisation between the business operating system and the web store. On a related note, the same or similar logic may be applied to other functions and areas of synchronisation.

The main idea for this type of synchronisation was based on practical experience and actual needs expressed by different business entities who have been using the Pantheon software. The idea was further explored in this thesis and it is our hope that it will become fully implemented sometime in the future.

Keywords: web store, business operating system, synchronisation

1 Uvod

1.1 Teoretično ozadje

Na področju računalniške znanosti se termin sinhronizacija uporablja v dveh podobnih, vendar zelo različnih konceptih: sinhronizacija procesov in sinhronizacija podatkov. Sinhronizacija procesov pomeni, da se mora v določeni časovni točki več različnih procesov uskladiti glede vrstnega reda izvajanja. Sinhronizacija podatkov pa se navezuje na idejo hranjenja več različnih koherentnih kopij podatkovnih setov oz. na vzdrževanje podatkovne integritete [25].

Sinhronizacija podatkov je kontinuiran proces kreiranja konsistence med izvornimi in ciljnimi podatki [26]. Je podmnožica replikacije [30], oboje pa pomeni prenos podatkov iz enega v drugi vir podatkov. Medtem ko replikacija pomeni prenos podatkov v celoti, sinhronizacija pomeni prenos podatkov v celoti ali pa samo deloma med različnimi viri. Ker je tematika diplomskega dela sinhronizacija podatkov med različnimi relacijskimi podatkovnimi bazami, bodo v nadaljevanju viri podatkov omejeni na relacijske baze.

Tako sinhronizacija kot replikacija se primarno izvajata iz dveh razlogov: boljša dostopnost in/ali hitrejši dostopi do podatkov. V obeh primerih bomo podatke zapisali v več različnih bazah, ki so tudi fizično locirane na ločenih strežnikih. S tem pridobimo več sistemskih virov, ki so na razpolago za manipulacijo nad podatki, hkrati pa povečamo dostopnost do podatkov, saj nedelovanje enega izmed virov ne pomeni tudi nezmožnosti dostopa do podatkov.

Sinhronizacija podatkov se lahko izvaja na več različnih načinov, katerega izberemo, pa je predvsem odvisno od tega, kaj želimo doseči. Eden od načinov sinhronizacije je lahko primeren v eni situaciji, medtem ko je isti način neprimeren v drugi situaciji. Razvojnik sistema je tisti, ki mora določiti, kako, kdaj in kaj se bo sinhroniziralo.

V grobem bi lahko sinhronizacijo z vidika dokončanja posodabljanja delili na sinhrono in asinhrono, druga večja delitev pa bi bila glede na število primarnih baz, se pravi baze, ki so na voljo tako za branje kot za pisanje. Znotraj teh dveh delitev je možno uporabiti različne tehnike sinhronizacije, ki jih bomo opisali kasneje (razdelek 1.1.6).

1.1.1 Sinhrona sinhronizacija

Sinhrona sinhronizacija pomeni takojšen prenos spremenjenih podatkov v sinhronizirane baze, sprememba pa se zgodi znotraj transakcije izvornega zahtevka [3]. To pomeni, da se sprememba podatkov sočasno aplicira na vse sinhronizirane baze, kar sicer pomeni realnočasovno posodabljanje vseh povezanih baz, vendar se kot posledica podaljša čas izvajanja operacije izvornega zahtevka, in sicer za čas trajanja posodobitve »najpočasnejše« baze. Na to lahko vplivajo hitrost mrežnih povezav, razpoložljivost sistemskih virov oz. kakršen koli drug dejavnik, ki vpliva na prenos podatkov med bazami. Druga zelo velika pomanjkljivost sinhrone sinhronizacije je nepredvidljivost napak oz. posledica le-te, saj se izvorna transakcija ne bo izvedla, če se zgodi napaka pri prenosu oz. potrditvi prenosa na kateri koli povezani bazi.

Poglejmo si praktični primer take napake: končni uporabnik želi v bazo, ki je sinhronizirana z ostalimi bazami, vnesti novega kupca. Uporabnik klikne na zapis novega kupca, vnese vse potrebne podatke in na koncu želi shraniti spremembe nazaj v bazo. Ob kliku na gumb shrani se izvede zapis podatkov v izvorno bazo, ki znotraj transakcije avtomatično izvede posodobitev ostalih baz. Ker pa je ena izmed povezanih baz nedostopna, se izvede preklic (rollback) izvorne transakcije in uporabniku se javi napaka, da vnos novega kupca ni mogoč niti na izvorni bazi.

Sinhrona sinhronizacija ima nekaj primerov uporabe, vendar izključno v nadzorovanem okolju, kjer so mrežne povezave stabilne.

1.1.2 Asinhrona sinhronizacija

Asinhrona sinhronizacija za razliko od sinhrone ne zahteva takojšnje posodobitve povezanih baz oz. vsaj ne znotraj izvorne transakcije. S tem se poveča hitrost izvorne operacije, saj se ne čaka na posodobitve ostalih baz, vendar pa to hkrati pomeni, da prihaja do začasne nekonsistence med podatki med bazami. Interval posodabljanja je kljub temu lahko zelo ažuren in tudi v tem primeru je možno doseči skoraj realnočasovno posodabljanje ostalih baz. Pozitivna stran tovrstne sinhronizacije je tudi manjši prenos podatkov, saj se prenašajo samo uspešno potrjene transakcije.

1.1.3 Ena primarna baza

Kot že omenjeno, se lahko sinhronizacije deli tudi glede na število primarnih baz. Najlažji način sinhronizacije med bazami je v primeru ene primarne baze in več replik. V tem primeru se vse spremembe izvajajo v primarni bazi in se naknadno sinhronizirajo v istem vrstnem redu

tudi v vse ostale povezane baze (kopije). Na tak način ne more prihajati do konfliktov, ki bi se lahko zgodili, če bi se podatki urejali istočasno v dveh oz. več različnih bazah.

1.1.4 Več primarnih baz

Bolj kompleksen primer sinhronizacije je več primarnih baz, v katere lahko uporabniki zapisujejo sočasno. Tovrsten sistem funkcionalne razporeditve baz nam po eni strani pohitri dostopnost do podatkov, saj se enaka kopija nahaja na več različnih lokacijah, po drugi strani pa tvegamo pojav nekonsistentnosti podatkov, do katerih pride zaradi konfliktov, ki se zgodijo, ko uporabniki v različnih bazah sočasno urejajo iste podatke. V tem primeru se konflikte najlažje prepreči prek sinhrone sinhronizacije, vendar pa je to, kot že rečeno, do neke mere tvegan način postavitve.

1.1.5 Napake in konflikti

V procesu sinhronizacije lahko pride do različnih napak, ki so lahko specifične glede na izbran način sinhronizacije oz. so napake bolj splošnega tipa, kot je npr. izpad mrežnih povezav. Konflikti so napake, ki so vezane na način sinhronizacije, do njih pa prihaja, ker se podatki »sočasno« urejajo v različnih bazah oz. zaradi zamika med spremembo podatka in njegovo sinhronizacijo v preostale baze. V primeru sinhrone sinhronizacije konflikti niso problematični, saj do njih ne more priti, lahko pa so problem v asinhroni sinhronizaciji. Razreševanje konfliktov je lahko do določene mere avtomatično ali pa ročno. Vse je odvisno od narave podatkov, ki jih obdelujemo. Avtomatično razreševanje konfliktov se izvaja večinoma glede na čas spremembe določenega podatka. Če se npr. isti podatek »sočasno« ureja v dveh različnih bazah, se upošteva tista sprememba, ki je bila kasnejša. S tem je seveda povezana tudi sinhronizacija časa med strežniki oz. operacijskimi sistemi, kjer so nameščene relacijske baze. V kompleksnejših sistemih, kjer je sinhroniziranih več relacijskih baz in so transakcije pogoste, pa lahko zaradi časovne cikličnosti zapisa podatkov prihaja do problemov, ki jih avtomatika ne more »pravilno« razrešiti.

1.1.6 Tehnike

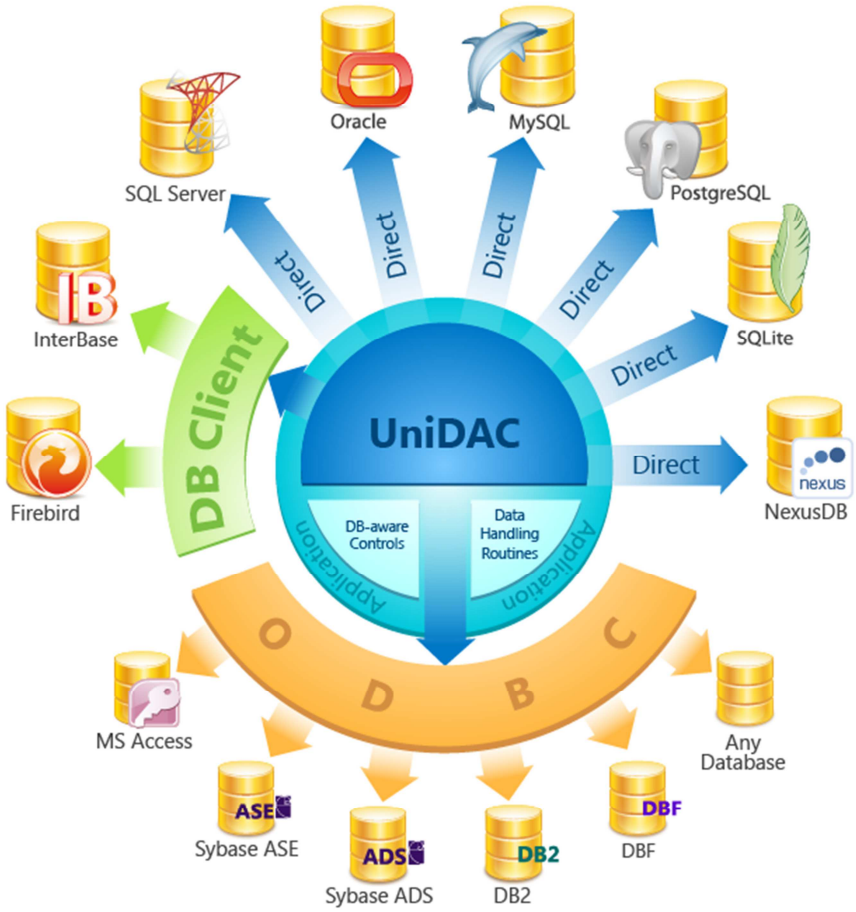
Obstaja več tehnik sinhronizacije in tudi razvijalci relacijskih baz ponujajo različne celostne rešitve za tovrstno problematiko. Poleg vgrajenih rešitev v relacijske baze obstajajo tudi zunanji ponudniki, ki z lastno programsko opremo nudijo rešitve za sinhronizacijo podatkov. Ne glede na rešitev pa se v osnovi sinhronizacija izvaja na podlagi dogodkov, zapisanih v dnevniku, ali pa s pomočjo prožilcev (triggerji) oz. je kombinacija obojega. Možna je tudi sinhronizacija na način direktne primerjave podatkov s pomočjo časovnih mehanizmov

(timestamp), vendar pa je to pri večjih podatkovnih bazah lahko preveč časovno zamudna operacija in posledično neuporabna.

Ko želimo sinhronizirati več različnih baz, je po navadi prvi korak začetna inicializacija baz, kar pomeni, da je stanje na vseh strežnikih enako oz. pred prvo izvedeno transakcijo obstaja identična kopija podatkov na vseh sinhroniziranih bazah (tisti del podatkov, ki se sinhronizirajo). Spremembe se v primeru sinhrone sinhronizacije lahko zapisujejo direktno v povezane baze prek prožilcev (triggerjev), v primeru asinhrone pa prek transakcijskega dnevnika dogodkov ali uporabniškega dnevnika, zapisanega v tabeli SQL, v katero se lahko zapisuje s pomočjo prožilcev (triggerjev). Zaradi razreševanja konfliktov je koristno, da se poleg sprememb vodi tudi čas spremembe, kar je mogoče prek časa, zapisanega v tabeli dnevnikov oz. prek časa spremembe podatkov, zapisanega v dodatnem polju v izvorni tabeli (timestamps).

1.2 Sinhronizacija heterogenih relacijskih baz

Z vidika sinhronizacije heterogenih sistemov (RDBMS) se je osnova izoblikovala že v zgodnjih 90. letih prejšnjega stoletja z uvedbo standarda ODBC (Open Database Connectivity) [13]. ODBC je aktualen še danes, vendar pa so ga nadgradile in nadomestile tudi druge tehnike, predvsem komponente, ki omogočajo poenoten dostop do več različnih baz podatkov s privzetimi gonilniki ciljne baze. Na Sliki 1.1 je prikazana shema možnih povezljivosti v različne RDBMS s komponentami UniDAC [9], ki se uporabljajo tudi v sinhronizaciji.



Slika 1.1: Povezljivost komponent UniDAC v različne RDBMS [9].

Sinhronizacija podatkov med različnimi informacijskimi sistemi prek mehanizma sinhronizacije relacijskih baz (homogenih ali heterogenih) lahko pomeni samo osnovo v celotnem procesu sinhronizacije informacij, saj se logika manipulacije nad podatki nahaja v višjenivojskih programskih jezikih. To pomeni, da sam prenos podatkov iz ene baze v drugo še ne pomeni dejanskega prenosa informacij iz enega informacijskega sistema v drugega. Da se podatek v ciljnem informacijskem sistemu pojavi kot informacija, je potreben še dodaten klic programske kode, ki je pisana v privzetem programskem jeziku ciljnega informacijskega sistema, ki bo podatke iz prehodnih oz. vmesnih tabel prenesel v lastne interne (SQL) strukture. Zaradi tega so v primeru različnih informacijskih sistemov trenutno v vzponu sinhronizacije prek spletnih storitev (web services), kjer je operacija sinhronizacije enonivojska oz. se podatek v ciljnem sistemu zapiše takoj v pravilne strukture SQL in ni potrebe po vmesnih oz. prehodnih tabelah.

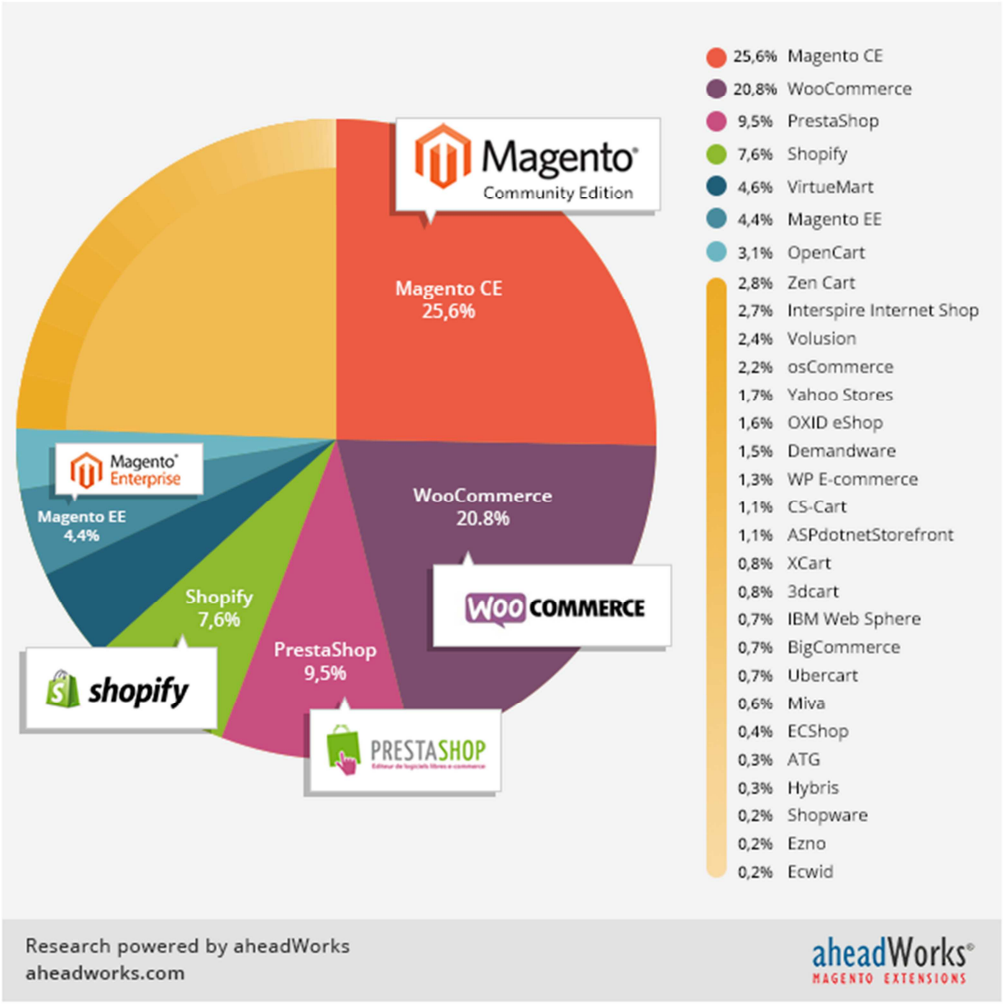
Sistemi za relacijske baze (RDBMS) večinoma omogočajo privzeto sinhronizacijo med lastnimi bazami (homogeno sinhronizacijo), medtem ko je zaradi kompleksnosti celostne

rešitve heterogena sinhronizacija prepuščena »zunanjim« programskim rešitvam. Tako lahko razvijalec sistema uporabi katero koli ogrodje, ki omogoča heterogen dostop, kot so ODBC [13], JDBC [27], firedac [28], unidac [29] ipd., končni uporabnik pa lahko izbira med različnimi programskimi rešitvami, kot so SymmetricDS [14], Talend [15] in Shadowbase [16].

1.3 Kratek opis ST

Včasih je bila praksa, da je vsak programer oz. podjetje razvilo lastno ST (enako je bilo tudi na ostalih področjih poslovanja), kar se je hitro izkazalo za slab poslovni model, predvsem z vidika stroškov vzdrževanja in razvoja. Internetne tehnologije se razvijajo hitro in tisti, ki jim niso sledili oz. se niso prilagajali času, so hitro ostali daleč zadaj. Tako je bilo tudi na področju ST, in sicer so se obdržale predvsem tiste, ki so imele dobro finančno zaledje ali pa dober razvojni model. S slednjim imamo v mislih predvsem odprtokodne sisteme, v obeh primerih pa je bila močna razvojna skupina ključ do uspeha.

Danes le redkokdo gradi ST od začetka. Na trgu prevladujejo predvsem odprtokodne rešitve in/ali rešitve SAAS. Graf na Sliki 1.2 prikazuje porazdelitev uporabe ST. Podatki so bili pridobljeni od enega izmed največjih spletnih orodij za analitiko spletnih strani Alexa (http://www.alexa.com/), obdelani pa s strani podjetja aheadWorks (top 1.000.000 strani). Omeniti je treba, da na spletu obstaja veliko podobnih analiz. Vsaka prikazuje podatke malo drugače, večinoma pa so podatki podobni podatkom na Sliki 1.2. Nekatere analize pa kažejo tudi večja odstopanja od prikazanih. Na žalost ne obstaja uradna entiteta oz. statistični urad, ki bi beležil tovrstne informacije.

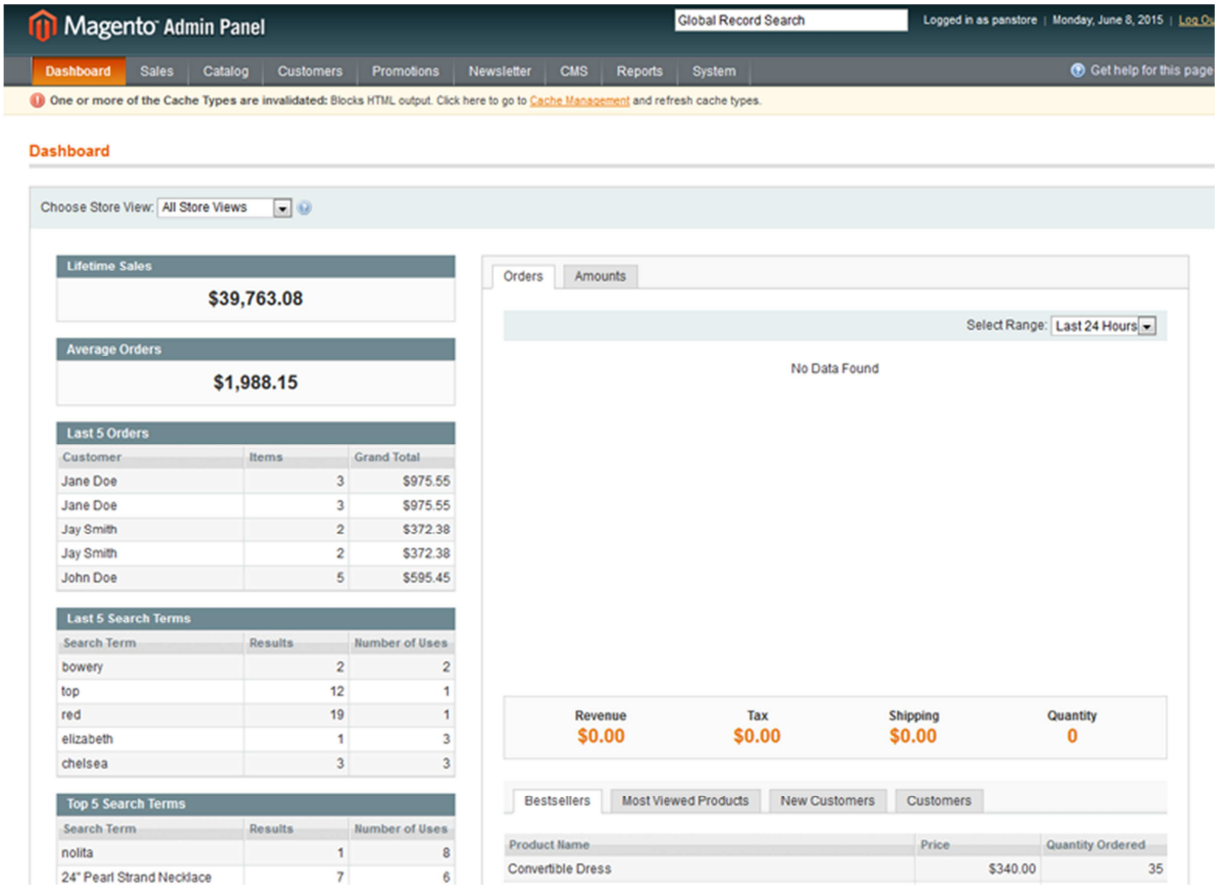


Slika 1.2: Primerjava globalnega tržnega deleža ST [25].

V Sloveniji je po naših izkušnjah stanje malo drugačno, saj ponudniki globalnih rešitev SAAS ne omogočajo slovenskega jezika in posledično ne oglašujejo na našem trgu, tako da tovrstne rešitve (razen ene slovenske – Shopamine) skoraj niso prisotne. Kljub vsemu menimo, da podani graf dokaj pravilno prikazuje slovensko stanje, če iz njega izvzamemo rešitve SAAS (woocommerce, shopify). Vodi Magento, sledita mu PrestaShop in OpenCart, v manjšem številu pa so prisotne tudi VirtueMart, osCommerce in ostale trgovine.

Sinhronizator trenutno podpira ST Magento (http://magento.com/), PrestaShop (https://www.prestashop.com/) in OpenCart (https://www.opencart.com/). Kasneje se bodo dodajale nove trgovine, predvsem odvisno od potreb končnih uporabnikov.

Na Sliki 1.3 je prikazana administratorska konzola ST Magento.



Slika 1.3: Administratorska konzola ST Magento.

1.4 Pregled trga in zahteve

Sinhronizacija v ožjem kontekstu diplomskega dela predstavlja programsko rešitev, potrebno za zagotovitev izmenjave podatkov med PIS in ST. V tem smislu je treba pogledati tako tehnološki del sinhronizacije kot tudi funkcionalne zahteve zanjo, ki skupaj definirajo potrebno tehnično rešitev.

Sinhronizacija med PIS in ST je potrebna tako zaradi zmanjševanja napak iz naslova ročnega dela kot tudi zaradi zagotavljanja hitrejših servisov ponudnikov na spletu. Kupci danes zahtevajo aktualne informacije o razpoložljivosti blaga/storitev, ažurne informacije o njihovem naročilu, hitro realizacijo naročila in personalizacijo. Vendar pa imajo različne spletne rešitve različne funkcionalne lastnosti in posledično različno podatkovno strukturo. Enaka situacija je tudi na strani PIS, kjer so si rešitve tehnično in tehnološko zelo raznolike.

Tako morajo ponudniki rešitev sinhronizacije ne samo usklajevati tehnični nivo izmenjave, temveč tudi razumeti vsebinski nivo rešitev PIS in ST, ker edino tako lahko zagotovijo smiselno integracijo podatkov na obeh straneh. Posledično je problematika sinhronizacije med

PIS in ST v praksi rešena na več načinov. Najpogosteje se ponuja individualne rešitve za potrebe sinhronizacije podatkov, kjer izvajalec skupaj z naročnikom najprej določi vsebinske in uporabniške zahteve sinhronizacije podatkov, preveri tehnične možnosti realizacije in predlaga arhitekturo rešitev ter na koncu razvije rešitev, ki deluje samo v informacijskem okolju izbranega uporabnika. Takšne rešitve bolj ali manj dobro delujejo, vendar so drage za razvoj. Učinkovitost rešitev je odvisna predvsem od tega, kako dobro je izvajalec razumel potrebe naročnika. Težava pa nastane pri vzdrževanju, saj je naročnik običajno vezan na enega samega ponudnika (t. i. »vendor lock in«). Vzdrževanje takšnih rešitev je tudi relativno drago, saj se stroški nadaljnjih prilagoditev zaradi sprememb v PIS ali ST delajo samo za eno stranko (ni ekonomije obsega). Po drugi strani pa takšne rešitve načeloma ponujajo zelo veliko prilagodljivost pri izvedbi naročnikovih zahtev, kar je omejeno praktično samo z višino vloženih finančnih sredstev naročnika in sposobnostmi izvajalca.

Druga situacija na trgu so majhna, fleksibilna podjetja, orientirana izključno na prodajne poti prek spleta, ki na strani PIS potrebujejo zelo malo vsebinske podpore. Tako se pogosto odločajo za ST, ki jim poleg funkcionalnosti spletnega trgovanja nudijo tudi osnovne funkcionalnosti, sicer vsebinsko primernejše za PIS. Ostale funkcionalnosti PIS prenesejo na zunanje izvajalce in na nivoju sodelovanja z njimi iščejo primerne oblike izmenjave podatkov. Običajno so tovrstni zunanji izvajalci kar računovodski servisi, ki svoje delo opravljajo na nivoju glavne knjige. Za svoje delo tako potrebujejo zgolj primerna poročila iz poslovanja, kot so različni obračuni izdanih in prejetih računov, stanje zaloge in bančni izpiski. Takšen pristop je lahko učinkovit za manjša podjetja, podjetja, ki se hitro spreminjajo, projektna podjetja in uporabnike, ki imajo zelo malo poslovnih zahtev.

Za drugačne uporabnike se na trgu pojavljajo tudi t. i. standardizirane rešitve, ki ponujajo različne rešitve za potrebe sinhronizacij, a so običajno nišno orientirane. To pomeni, da so standardizirane za povezavo med točno določenim PIS in točno določenimi ST. Ekonomija obsega se pri takšnih rešitvah že lahko razvije in so zato lahko stroškovno bolj učinkovite. Po drugi strani pa takšne rešitve po navadi ne dosegajo visoke stopnje fleksibilnosti in prilagodljivosti uporabnikovim posebnostim. V svetu hitrega razvoja in visoke stopnje spremenljivosti trgov imajo tako samo omejen uspeh.

1.5 Konkurenca

Konkurenco na trgu predstavljajo različni ponudniki z različnimi rešitvami in predvsem različni poslovni pristopi k tej problematiki, ki jih uporabniki, običajno slabo informirani o možnostih, iščejo sami. Pri raziskovanju konkurence smo pogledali dva nivoja: regionalni

nivo, ki obsega države nekdanje Jugoslavije, in evropski nivo, ki predstavlja EU in prej omenjeno regijo skupaj.

V regiji je konkurenca slabo razvita. Prednjačita dve obliki konkurence, ki pa zaradi relativno slabe razgledanosti potencialnih uporabnikov nista zanemarljivi. Na trgu srednje velikih podjetij/rešitev prednjači konkurenca v obliki samostojnih podjetij, ki ponujajo individualne rešitve razvoja sinhronizatorja po potrebi naročnika. Ti ponudniki so značilno orientirani na izbrani PIS in nudijo storitve razvoja rešitev po potrebi za znanega naročnika. Značilno imajo tudi monopol nad nudenjem teh storitev na trgu izbranega PIS.

Druga oblika konkurence so informacijsko relativno dobro podprti računovodski servisi, ki z diverzifikacijo svoje ponudbe skušajo povečati obseg svojega poslovanja. Tisti, ki imajo bodisi kritičen obseg informacijsko-tehnološkega znanja ali pa razvite dobre partnerske odnose z informacijsko-tehnološkimi podjetji, svojim strankam nudijo storitve, ki pogosto presegajo knjigovodstvo in računovodstvo. Pogosto majhnim podjetjem ponujajo celo t. i. »outsourcing« poslovnih funkcij ter informacijske rešitve za podporo procesom sinhronizacije podatkov. Tako nastajajo rešitve uvoza podatkov v sisteme PIS, ki jih uporabljajo računovodski servisi za opravljanje svojih storitev, s katerimi se olajša prenos podatkov iz točke zajema (ST) do PIS. Te rešitve običajno obsegajo uvoz podatkov iz datotek XLS, CSV, TXT ali XML s standardizirano strukturo. Razvoj takšnih rešitev ni zahteven, hkrati pa imajo računovodski servisi možnost realizacije ekonomije obsega.

Širše v evropskem okolju obstajajo tudi nekatere standardizirane rešitve, ki za različne izbrane ST ponujajo možnost prevzema podatkov, torej omogočajo nekakšen API do podatkov v ST. Opaziti je celo trend razvoja funkcionalnosti PIS za potrebe poslovodenja na blagovnem delu, ki se ponujajo kot nadgradnje funkcionalnosti [31]. Trenutno je ta trend zaznati na področju večjih sistemov spletnega trženja, kot je recimo Amazon, in pa na področju ST z višjo stopnjo tržne penetracije, kot je recimo Magento. Slabi strani teh rešitev sta, da le deloma zagotavljajo informacijsko podporo ostalim procesom, ki jih sicer pokrivajo PIS, in da je fleksibilnost tovrstnih rešitev nizka. Zaradi nuje ponudnikov takšnih rešitev je odnos slednjih običajno »vzemi ali pusti«. Iz vidika slovenskih kupcev pa takšne rešitve tudi ne ponujajo zaradi zakonodajnih specifik oz. t. i. »lokalizacije« [31].

V prejšnjem odstavku omenjene rešitve so običajno realizirane kot spletne aplikacije. Glede na izvor podjetij, ki jih običajno ponujajo, je to tudi razumljivo, saj jih večina izhaja iz branže razvoja ST. To ima dobro plat v smislu neodvisnosti od operacijskega sistema in mobilnosti. A pogosto so te rešitve zaradi tržnega modela SAAS ponujene kot storitve v oblaku, kar pa poleg stroškovnih ugodnosti za srednje velika in velika podjetja prinaša tudi tveganje. Slednji

namreč niso naklonjeni rešitvam, kjer bi se poslovni podatki hranili izven obsega njihovega obvladovanja.

1.6 Povzetek diplomskega dela

Diplomsko delo predstavlja enega izmed možnih načinov sinhronizacije različnih podatkovnih sistemov tako na nivoju relacijskih baz kot na nivoju ciljne aplikacije. Na eni strani se nahaja PIS, ki ga večina podjetij uporablja kot jedro lastnega poslovanja, na drugi strani pa ST, ki služijo predvsem predstavitvi prodajnega programa in omogočajo avtomatiziran zajem naročil kupcev.

Opisana je dvosmerna sinhronizacija ključnih funkcionalnosti, potrebnih za prodajo prek spleta, in sicer sinhronizacija naročil, sinhronizacija zaloge in sinhronizacija cen.

V drugem poglavju je opisan koncept sinhronizacije. Sinhronizacija se je v osnovi realizirala dvonivojsko. Na najnižjem (SQL) nivoju se je realizirala potreba po sinhronizaciji podatkov med heterogenimi podatkovnimi bazami in hkrati bolj generičnem pristopu k sinhronizaciji. Na aplikativnem nivoju oz. z višjenivojskim programskim jezikom pa se je realizirala potreba po integraciji podatkov v ciljni informacijski sistem (ST), hkrati pa tak pristop k sinhronizaciji pomeni tudi praktično neomejeno razširljivost in modularnost celotnega sistema. Kot rezultat se je kreiralo več različnih izvršilnih datotek na strani PIS in več skript, razvitih v programskem jeziku PHP, na strani ST.

V tretjem poglavju so predstavljena razvojna orodja, ki so bila uporabljena za razvoj rešitve sinhronizacije med PIS in ST.

Četrto poglavje opisuje vizualne komponente sinhronizacije, ki so potrebne za namestitev, izvajanje, nastavitve in pregled procesa sinhronizacije. Na strani PIS so bile komponente pisane za operacijski sistem Windows, na strani ST pa komponente predstavljajo izvršljive skripte PHP. Zaradi lažje predstave podatkovnega modela je najprej predstavljen vizualni del komponent.

Peto poglavje opisuje enega izmed najpomembnejših področij naše rešitve, in sicer podatkovni model oz. nivo SQL sinhronizacije. Razložimo logiko poimenovanja struktur SQL in funkcionalno uporabo tabel SQL, ki jih prikažemo tudi s pomočjo diagramov ER. Za lažjo predstavo opišemo logiko delovanja sinhronizacije med različnimi sistemi tudi s pomočjo komunikacijskih diagramov. Na koncu se dotaknemo še vodenja dnevnikov, kjer je podrobno zapisano, kaj se dogaja v procesu sinhronizacije, in sicer tako uspešno izvedene akcije kot tudi napake.

Šesto poglavje je prav tako eno izmed ključnih področij delovanja rešitve kot celote, in sicer opisujemo postopek namestitve komponent tako na strani PIS kot na strani ST. Poleg avtomatizirane namestitve sinhronizacije opišemo tudi možnosti uporabniških nadgradenj privzetih funkcionalnosti.

V sedmem poglavju opišemo zastavljene mehanizme za izvajanje postopka testiranja. Tako kot v procesu namestitve se tudi tukaj izkaže heterogenost sistemov kot velik izziv za postavitev enotnega procesa testiranja sinhronizacije.

V osmem poglavju se dotaknemo področja dokumentacije. Ker je naša želja, da bi se sinhronizator funkcionalno širil tudi s pomočjo »zunanjih« ponudnikov, se pravi tistih, ki skrbijo za informacijski sistem končnega uporabnika, je dobra dokumentacija ključnega pomena.

Zadnje poglavje predstavlja sklepne ugotovitve in razmišljanje, kako naj bil začrtan razvoj v prihodnosti.

2 Sinhronizacija

2.1 Sinhronizator

Sinhronizator smo poimenovali programsko rešitev, ki izvaja funkcijo sinhronizacije med PIS in ST. Osnovno vodilo pri razvoju sinhronizatorja je bil razvoj standardizirane rešitve, ki bi izpolnjevala osnovne pogoje za komercialne modele ekonomije obsega in bi tako na trg prinesla cenovno dostopno ter tehnološko dovršeno rešitev. Rešitev bi morala biti sposobna vzpostavitve komunikacije med različnimi podatkovnimi bazami. Hkrati mora biti takšna rešitev dovolj fleksibilna, da lahko z minimalnimi dodatnimi stroški implementiramo nove vsebinske zahteve sinhronizacije podatkov. Generalna raba sinhronizatorja se na začetku pričakuje za sinhronizacijo med PIS in ST.

Konceptualno predstavlja sinhronizator samostojno entiteto v celotnem procesu sinhronizacije in je tehnološko neodvisen od informacijskega sistema oz. podatkovne baze tako na eni kot na drugi strani. Neodvisnost entitete je realizirana predvsem s pomočjo ločenih podatkovnih struktur, ki služijo hrambi podatkov v celotnem procesu sinhronizacije. Zaradi načina hrambe podatkov oz. časa trajanja te hrambe se (nekaterim) tabelam sinhronizatorja lahko reče tudi prehodne tabele. Iz vidika sinhronizatorja prehodne tabele predstavljajo osnovne podatkovne strukture, ker pa je sinhronizator vedno treba gledati v kontekstu obeh povezanih informacijskih sistemov (PIS in ST), je iz tega vidika treba gledati tudi omenjene tabele.

Kot primer si poglejmo proces sinhronizacije zaloge, kjer se ob spremembi zaloge v PIS prek prožilca ta podatek zapiše v tabelo, definirano s strani sinhronizatorja. Tam podatek čaka na prenos in v trenutku, ko sinhronizator uspešno posodobi zalogo na strani ST, bo podatek iz tabele izbrisan. Ravno zaradi te začasnosti zapisa imenujemo tabelo prehodna. Enaka oz. podobna logika se uporablja pri vseh spremembah, ki se zgodijo na strani PIS.

Shematsko je sinhronizator sestavljen iz izvršilnih datotek (Delphi [22]), ki so v večini primerov nameščene na strani PIS (lahko tudi na drugem računalniku) iz tabel in pogledov na strani relacijske podatkovne baze in iz skript (PHP [34]) na strani ST. Vse tri sisteme med seboj povezujejo samo lastne podatkovne strukture sinhronizatorja. Celoten koncept sinhronizatorja je z vidika generičnosti in modularnosti zasnovan ravno na standardiziranih podatkovnih strukturah (tabele in pogledi), se pravi strukturah, ki so enake pri vseh PIS in ST.

Možne in tudi predvidene so tudi nestandardizirane nadgradnje podatkovnih struktur, prek katerih lahko hitro in preprosto zadostimo različnim uporabniškim zahtevam.

Obstaja veliko tehnoloških in tehničnih načinov sinhronizacije med PIS in ST, vendar bi lahko realizacijo večine omenjenih načinov izvedbeno razdelili v tri skupine:

- sinhronizacija s pomočjo prenosa datotek (XML, CSV, XLS ipd.);
- sinhronizacija na nivoju relacijske baze;
- sinhronizacija s pomočjo spletnih servisov.

2.1.1 Sinhronizacija s pomočjo prenosa datotek

Sinhronizacija s pomočjo prenosa datotek predstavlja enega izmed najlažje izvedljivih načinov. Tako izvoz podatkov v datoteko kot uvoz podatkov iz datoteke predstavljata neodvisen korak v celotnem procesu sinhronizacije, poleg tega pa so lahko datoteke univerzalen način za sporazumevanje med dvema razvijalcema. Z univerzalnim načinom imamo v mislih predvsem dostopnost do podatkov, shranjenih v datoteki, če se le uporabi dovolj splošno podprt format datoteke. To je pomembno predvsem zato, ker je omogočen neodvisen razvoj implementacije tako izvoza kot uvoza podatkov, tehnološki in tehnični pristop pa je prepuščen razvijalcu, ki se bo odločil glede na svoje zmožnosti in znanje. Po drugi strani pa so lahko datoteke, predvsem z vidika njihovega prenosa, okorne, saj je za to potreben ločen protokol, ki predstavlja nov korak v procesu sinhronizacije, kar pa vpliva tako na večjo možnost dogodka napake kot tudi na hitrost prenosa podatkov med obema informacijskima sistemoma. Najpogosteje se za ta korak uporablja protokol FTP.

2.1.2 Sinhronizacija na nivoju relacijske baze

Ta način je bistveno boljša izbira, če sinhroniziramo podatke med dvema relacijskima bazama, ki se lahko neposredno povežeta, in imamo do njiju neposreden dostop oz. le-tega lahko pridobimo. V tem primeru bi lahko govorili že o skoraj privzetem načinu sinhronizacije, še posebej zato, ker nam razna ogrodja, kot je npr. ODBC, omogočajo povezovanje med heterogenimi relacijskimi bazami. Za razliko od sinhronizacije s pomočjo datotek v tem primeru ni potrebe po shranjevanju podatkov v »zunanji« vir, ampak jih lahko zapišemo neposredno iz izvorne v ciljno tabelo SQL. Kot vemo, predstavlja tabela v relacijski bazi osnovni element za shranjevanje podatkov. Tudi z vidika performanc in možnosti dogodka napake oz. – mogoče še pomembnejše – korekcije te napake je veliko lažje realizirati sinhronizacijo neposredno na nivoju dveh relacijskih baz. Potrebujemo le ustrezen gonilnik,

ki omogoča prenos podatkov iz ene relacijske baze v drugo. Največje pomanjkljivosti te vrste sinhronizacije, v primerjavi z ostalima dvema možnostma, so skrb za varnost podatkov, potrebna namestitev ustreznega gonilnika za komunikacijo s ciljno bazo in aplikacija podatkov v ciljni informacijski sistem. Tako menimo predvsem z vidika, da ima večina aplikacij, ki v ozadju uporabljajo relacijsko bazo, uvoz podatkov realiziran prek aplikativne logike, ki ni realizirana neposredno na nivoju relacijske baze. Relacijska baza je velikokrat samo orodje za shranjevanje podatkov in ima zelo omejene možnosti nad aplikativno manipulacijo podatkov, ki pa je omogočena oz. realizirana v višjenivojskem programskem jeziku (Delph [22], .NET [32], Java [33], PHP [34] itd.).

2.1.3 Sinhronizacija s pomočjo spletnih servisov

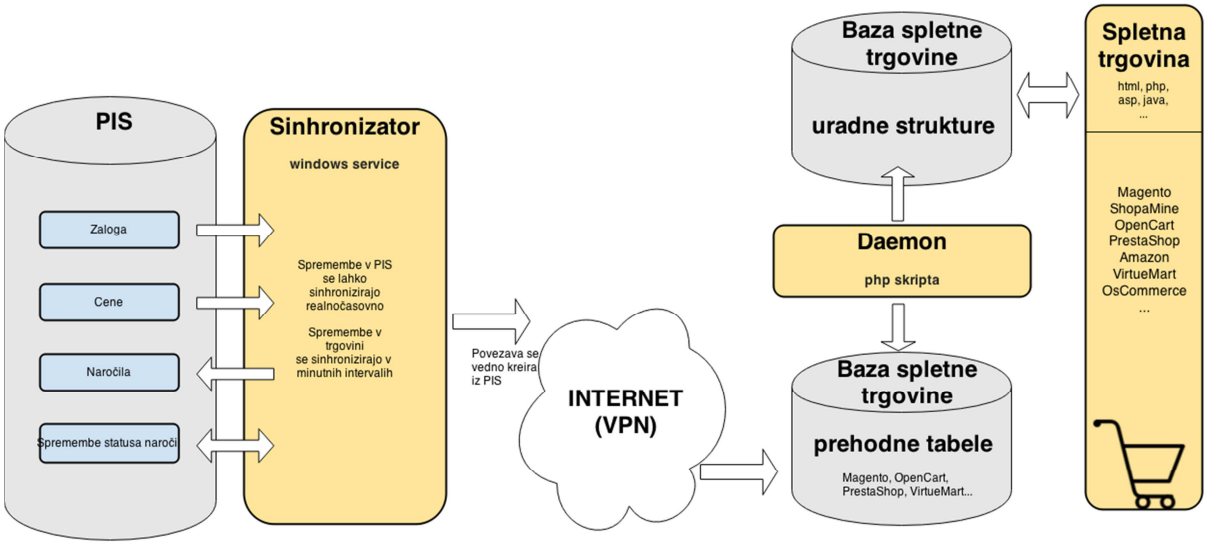
Spletni servisi so v svojem jedru delovanja zelo podobni sinhronizaciji s pomočjo datotek, saj se dejansko v večini primerov za izmenjavo podatkov uporablja ravno notacija XML, ki je identična tudi pri kreiranju datoteke. Razlika je samo v tem, da v primeru spletnih servisov ne shranjujemo v datoteko, ampak v podatkovni vir, ki ga določa spletni servis. Seveda pa spletni servisi omogočajo veliko več funkcionalnosti in strukturiranega načina komunikacije med klientom in strežnikom. Uporabljamo jih predvsem v primerih, ko želimo na standardiziran način ponuditi storitve/podatke čim širšemu krogu uporabnikov, pri tem pa uporabiti čim manj oz. nič človeških virov za komunikacijo usklajevanja procesa sinhronizacije. Po sami naravi je spletni servis orodje, ki omogoča avtomatiziran nivo komunikacije z razvijalcem na drugi strani. Če predpostavljamo uporabo standardiziranih tehnik (REST) in dobro dokumentacijo, je lahko razvoj komunikacije med sistemoma tako na eni kot na drugi strani neodvisen in samostojen. Tak način pa načeloma pomeni tudi večji vložek v začetni razvoj v primerjavi s prej omenjenima načinoma.

V sinhronizatorju se pogojno uporabljata dve izmed zgoraj naštetih tehnik (opisane v razdelkih 2.1.1 - 2.1.3). V vsakem primeru se uporablja neposredna sinhronizacija med relacijskima bazama (razdelek 2.1.2), kar nam omogoča heterogenost sinhronizacije, opcijsko pa se koristi tudi spletne servise (razdelek 2.1.3), če jih informacijski sistem omogoča, oz. kar nativne klice API ciljnega informacijskega sistema. Drugače povedano: v samem jedru sinhronizator omogoča sinhronizacijo med izvorno in ciljno relacijsko bazo, od razvojnika pa je odvisno, kako bo te podatke apliciral v končni oz. jih prebral iz izvornega informacijskega sistema. Razlog za tak način delovanja je potreba po podpori čim večjemu številu različnih informacijskih sistemov, kar najlažje dosežemo prek visoko standardiziranega in strukturiranega jezika SQL ter z željo po modularnosti sinhronizacije, kjer je integracija v informacijski sistem realizirana prek (neodvisnega) sistema vtičnikov (dodatkov »add-on«).

Obstaja pa manjša razlika v implementaciji sinhronizacije glede na smer prenosa oz. glede na področje delovanja, ki ga želimo sinhronizirati. V primeru, da sinhroniziramo iz PIS v ST, se spremembe zajema s pomočjo prehodnih tabel, v katere se zapisuje prek prožilcev. V primeru, ko sinhroniziramo iz ST v PIS (spletna naročila), pa uporabljamo samo standardizirane poglede. Različen koncept oz. metodologija se uporablja predvsem zaradi kompleksnosti zajema sprememb nad sinhroniziranimi podatki. V primeru sinhronizacije naročil je treba spremljati spremembo samo v enem stolpcu tabele in še to nas zanimajo samo novi zapisi, se pravi nova naročila. Trgovine namreč ne dopuščajo popravkov nad že vnesenimi naročili oz. vsaj ne nad tistimi podatki, ki se jih prenaša v PIS (izjema je le status naročila). Poleg tega pa hitrost zaznane spremembe ni bistvenega pomena in stavek SQL »*select * from naročila where id_naročila > id_naročila_pis_maks*« vsakih 5 minut zadostuje potrebam po evidentiranju novih naročil in hkrati ne obremenjuje trgovinskega sistema.

Na strani PIS so procesi zajemanja sprememb kompleksnejši. Poleg tega je večja potreba po realnočasovni evidenci in aplikaciji sprememb, zato ne moremo uporabiti enakega pristopa kot na strani ST. Sprememba zaloge se privzeto preverja vsake 3 sekunde. Poizvedba *»select * from zaloga where datum_spremembe > zadnji_datum_prenosa«* bi na večjih sistemih z več 10.000,00 oz. celo več 100.000,00 artikli močno obremenjevala sistem. Za razliko od ST, kjer je stolpec »id_naročila« večinoma primarni ključ in posledično indeksiran, »datum_spremembe« ni niti obvezen oz. pogojno indeksiran stolpec, kar še dodatno onemogoča pristop brez uporabe prožilcev. Enako velja tudi za sinhronizacijo cen, kjer je kompleksnost še malo večja, saj artikel v PIS vsebuje več različnih cen in zelo veliko dodatnih informacij o artiklu, zato bi bilo samo z uporabo polja »datum_spremembe« skoraj nemogoče učinkovito sinhronizirati spremembe iz PIS v ST.

2.2 Shema sinhronizacije



Slika 2.1: Shema sinhronizacije.

Shema na Sliki 2.1 prikazuje podatkovno pot in elemente, ki nastopajo v procesu sinhronizacije. Začne se v podatkovni bazi PIS, od tu naprej pa se podatki premaknejo prek prehodnih tabel s pomočjo sinhronizatorja na stran ST. V ST jih prevzame demon (PHP), ki skrbi za njihovo aplikacijo v podatkovne strukture ST. Pot lahko poteka tudi v obratni smeri, vendar pa je zaradi varnosti povezava vedno inicializirana s strani PIS (ST ne sme imeti direktnega dostopa do podatkovne baze PIS, saj je lahko narava podatkov, ki se hranijo v PIS, zelo občutljiva in bi se tako po nepotrebnem povečevalo tveganje nepooblaščenega dostopa do podatkov).

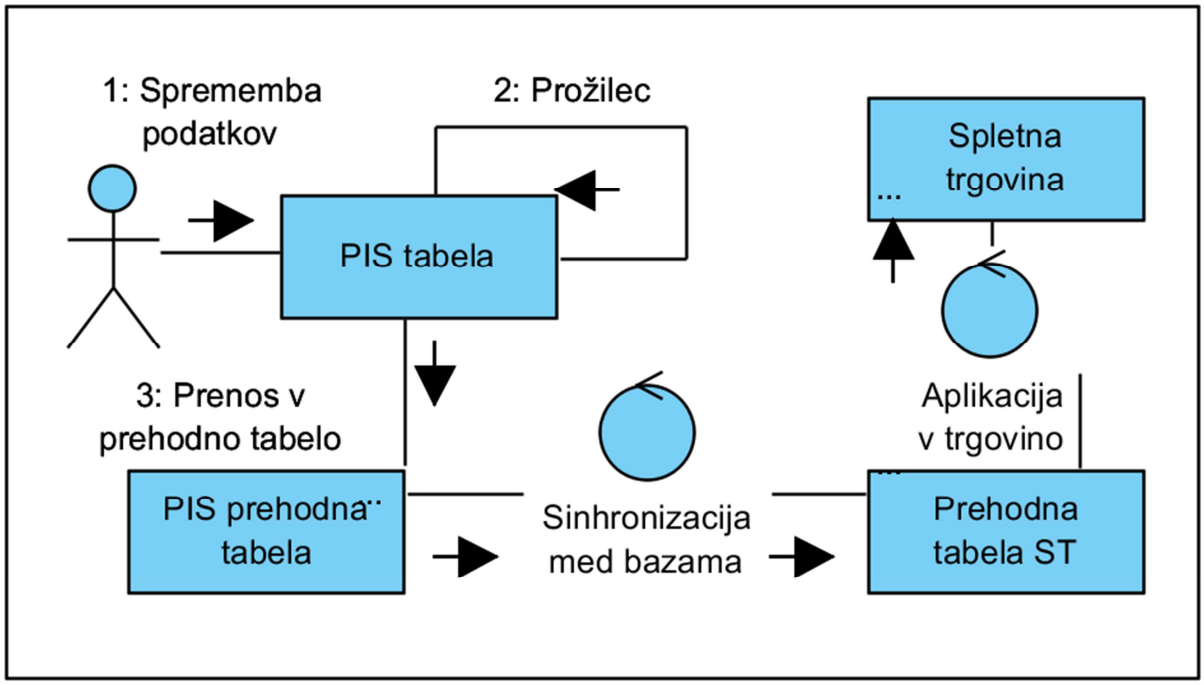
2.3 Modularnost sinhronizatorja

Modularnost je omogočena predvsem zaradi uporabe prehodnih tabel in neodvisne aplikacije podatkov tako na strani PIS kot na strani ST. Aplikacija podatkov iz prehodnih tabel v informacijski sistem se dogaja na nivoju ciljnega informacijskega sistema (ST), kar pomeni, da je omogočena uporaba nativnih klicev sistema API. Poleg tega so predvideni uporabniški posegi v privzete strukture SQL (procedure, funkcije, pogledi, tabele), kar pomeni, da lahko končni uporabnik določeno funkcionalnost prepiše glede na lastne potrebe in ta funkcionalnost se bo ohranila tudi po nadgradnji sinhronizatorja (po principu dodatkov »add-on«).

Prenos podatkov je podprt tako v smeri ST kot v smeri PIS in poteka po sledeči shemi:

PIS <=> prehodne tabele <=> servis Windows <=> prehodne tabele <=> demon php/java/ipd. <=> ST.

2.3.1.1 Shema za prenos iz PIS v ST

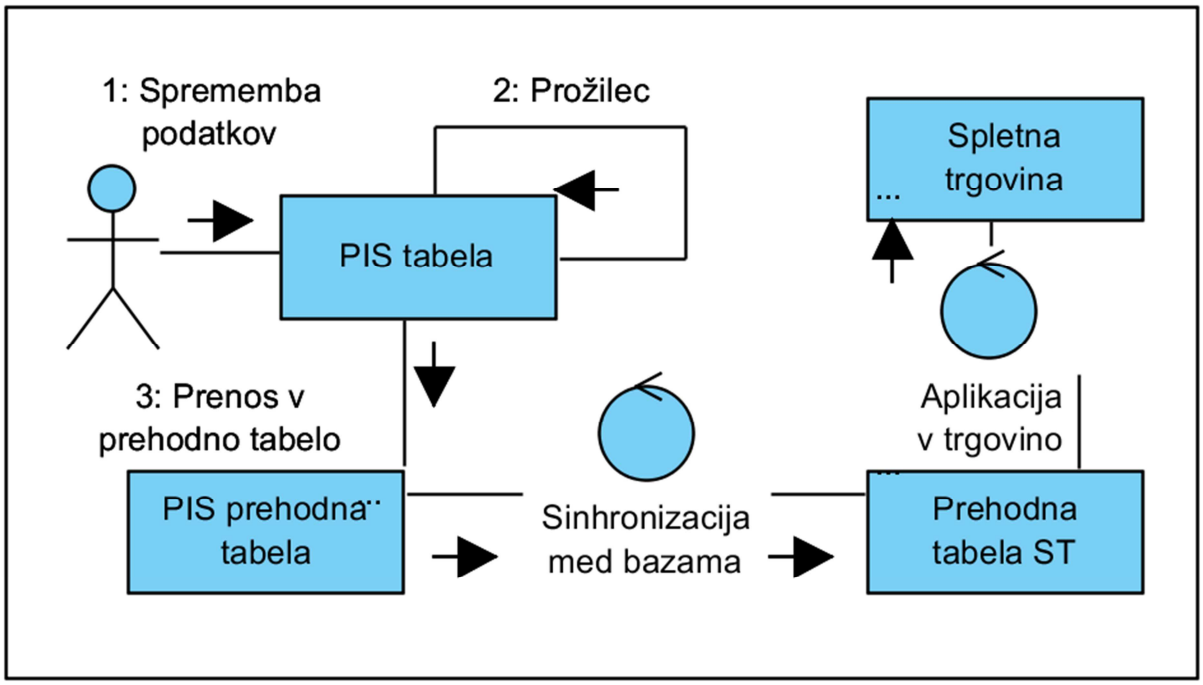


Slika 2.2: Poenostavljena shema za prenos iz PIS v ST.

Kot je razvidno s Slike 2.2, v procesu sinhronizacije nastopata dve kontrolni entiteti, ena na strani PIS in ena na strani ST. Ti dve entiteti skrbita za aplikacijo podatkov v podatkovne strukture informacijskega sistema in delujeta kot samostojni enoti.

V primeru prenosa podatkov iz PIS v ST morajo biti zagotovljeni hitri prenosi, saj obstaja velikokrat težnja po skoraj realnočasovni sinhronizaciji podatkov oz. je le-ta ključnega pomena. Za prenos podatkov v čakalno vrsto sinhronizacije skrbijo sprožilci, ki morajo biti pisani preprosto in delovati zelo hitro. Pod nobenim pogojem se ne sme zgoditi, da bi bila stabilnost PIS zaradi dodatnih prožilcev omajana oz. da bi se uporabniška izkušnja občutno podaljšala zaradi prekompleksnih operacij v sprožilcih (sprožilci se prožijo znotraj uporabniške seje, kar pomeni, da se čas izvajanja operacije končnega uporabnika podaljša za čas izvajanja prožilcev, ki so namenjeni sinhronizaciji s ST).

2.3.1.2 Shema za prenos iz ST v PIS



Slika 2.3: Poenostavljena shema za prenos iz ST v PIS.

V primeru sinhronizacije iz ST se ne uporablja prožilcev (triggerjev), ker hitrost zaznane spremembe ni tako bistvenega pomena kot pri prenosu v obratno smer. Poleg tega se sinhronizator nahaja na strani PIS, kar pomeni bistveno hitrejšo komunikacijsko pot (pri spremembi določenih podatkov v PIS je ključnega pomena, da se zgodi sinhronizacija skoraj realnočasovno (zaloga), medtem ko hitrost prenosa spletnih podatkov v PIS (naročila) ni tako pomembna).

2.4 Generičnost sinhronizatorja

Generičnost je potrebna predvsem na nivoju sinhronizacije baz SQL med PIS in ST, kar lahko dosežemo s pomočjo komponent Delphi za enotno povezovanje na različne baze podatkov in uporabe standardiziranih struktur SQL. To nam omogoča prenos podatkov iz baze SQL v tranzicijski objekt Delphi, ki se mu reče »dataset«, s katerim lahko manipuliramo identično in ne glede na to, s katero bazo podatkov smo povezani. Kljub vsemu je treba biti pazljiv pri prenosu podatkov iz strežnika SQL v »dataset«, saj je treba pisati tako generično kodo, da je sintaktično pravilna na vseh podprtih strežnikih SQL (možna so določena odstopanja, vendar

se z vsakim odstopanjem poveča kompleksnost kode, kar pomeni težje vzdrževanje in nadgradnje). Večinoma se uporablja koda tipa

»*select * from prehodna_tabela*«,

vsa ostala manipulacija nad podatki pa se izvaja v okolju Delphi.

3 Razvojna programska orodja

Ker je o razvojnih orodjih napisano že veliko in ker so le-ta zelo dobro poznana, bomo to tematiko predstavili na hitro, opisi orodij pa so povzeti iz tujih virov. Pri razvoju sinhronizatorja so bila uporabljena razvojna okolja, ki so predstavljena v nadaljevanju.

3.1 Microsoft SQL Server Management Studio

SSMS je integriran produkt, ki služi za dostop, nastavitve, urejanje, administriranje in razvijanje vseh komponent strežnika SQL. SSMS združuje tako grafične pripomočke kot tudi tekstovne urejevalnike besedil za dostop do strežnika SQL tako za razvojnike kot tudi za administratorje ne glede na njihovo stopnjo znanja.

SSMS združuje lastnosti produktov »Enterprise Manager«, »Query Analyzer« in »Analysis Manager«, ki so bili vključeni v prejšnjih verzijah strežnika SQL v eno skupno programsko rešitev. SSMS deluje z vsemi komponentami strežnika SQL, kot so orodja za poročanje (Reporting Services) in integracijska orodja (Integration Services). Razvojniki tako dobijo poznano okolje, administratorji pa enotno programsko opremo, ki združuje preprosta grafična orodja in veliko možnosti za skriptno podporo [18].

SSMS smo uporabili pri razvoju podatkovnega modela na strani PIS (Pantheon), ki uporablja Microsoftov strežnik SQL kot RDBMS.

3.2 Microsoft SQL Server

Microsoftov strežnik SQL je sistem za upravljanje relacijskih podatkovnih baz, ki ga je razvilo podjetje Microsoft. Njegova primarna funkcija je hranjenje in obdelava podatkov glede na zahteve ostalih programskih rešitev, ki so lahko nameščene na enak računalnik kot strežnik SQL ali pa na drug računalnik v omrežju (vključno z internetom).

Microsoft ima v ponudbi vsaj ducat različnih izdaj strežnika SQL za različne ciljne skupine in za nivoje uporabe, in sicer od nezahtevnih aplikacij, nameščenih na enem računalniku, do ogromnih internetnih aplikacij, ki jih uporablja veliko sočasnih uporabnikov.

Strežnik uporablja kot primarni jezik sintakso T-SQL ali ANSI SQL [19].

Microsoftov strežnik SQL uporablja večina PIS, s katerimi želimo vzpostaviti sinhronizacijo, in sicer tudi Pantheon, ki je služil kot osnova za izvedbo podatkovnega modela na strani PIS.

3.3 MySQL

MySQL je sistem za upravljanje relacijskih podatkovnih baz (RDBMS). V juliju 2013 je bila to druga svetovno najbolj razširjena baza podatkov (RDBMS) in najbolj razširjen odprtokodni RDBMS. Baza je poimenovana po hčerki My soustanovitelja Michaela Wideniusa. SQL je kratica, ki pomeni strukturno-poizvedbeni jezik. Razvojna koda programske rešitve MySQL je na voljo pod pogoji GNU General Public Licence, prav tako pa je na voljo pod različnimi zaščitenimi oz. lastniškimi licencami. Lastnik in sponzor MySQL je bilo včasih švedsko podjetje MySQL AB, sedaj pa je v lasti podjetja Oracle. Na voljo je tudi nekaj plačljivih različic, ki ponujajo dodatne funkcionalnosti.

MySQL je popularna izbira baze za uporabo v spletnih aplikacijah in je ena izmed glavnih aplikacij v zelo razširjeni uporabi odprtokodnega okolja LAMP in ostalih okoljih AMP. LAMP je kratica za »Linux, Apache, MySQL, Perl/PHP/Python« [20].

MySQL uporablja večina ST kot svoj RDMBS, in sicer tudi tiste, s katerimi smo mi vzpostavili sinhornizacijo.

3.4 dbForge Studio za MySQL (standard)

dbForge Studio je univerzalen klient za operacijski sistem Windows, ki omogoča dostop razvojnikom in administratorjem, da med drugim kreirajo, izvajajo, razvijajo, razhroščujejo in avtomatizirajo rutine na podatkovni bazi MySQL v priročnem okolju.

Orodje ponuja tudi možnosti za primerjavo, sinhronizacijo, varnostno kopiranje podatkov z vgrajenim časovnikom (scheduling), analiziranje podatkov in orodja za poročanje. 15.000 uporabnikov uporablja orodje za urejanje, vzdrževanje in nadziranje baze podatkov MySQL [21].

Obstaja več različic orodja, ki ponujajo različne funkcionalnosti (brezplačna, standard, profesionalna). Glede na naše izkušnje je to najboljše orodje za delo z bazo podatkov MySQL. Že brezplačna različica zadosti večini potreb za razvojno delo z omenjeno bazo.

Podobno kot SSMS na strani PIS smo na strani ST uporabili dbForge Studio za manipulacijo nad podatkovnimi strukturami in podatki, shranjenimi v bazi podatkov MySQL.

3.5 Embarcadero Delphi

Embarcadero Delphi je integrirano razvojno okolje (IDE) za konzolne, namizne, grafične, spletne in mobilne aplikacije.

Prevajalnik Delphi uporablja lastno programsko kodo, imenovano objektni Pascal (različica Pascala), za generiranje binarne kode za naslednje platforme: 32- in 64-bitni Windows, 32-bitni Mac OS X, iOS in Android.

Delphi je bil v osnovi razvit s strani podjetja Borland kot »rapid application development tool« za operacijski sistem Windows in se je štel kot naslednik Borland Pascala. Delphi in »C++ Builder« sta uporabljala veliko skupnih jedrnih komponent, kot sta IDE in VCL, vendar sta bila do izida različice RAD Studio 2007 ločena produkta [22].

Embarcadero Delphi smo uporabili kot razvojno okolje za razvoj glavnih komponent sinhronizatorja, ki se jih namesti na operacijski sistem Windows. Komponente, kreirane s pomočjo okolja Delphi, uporabljamo kot uporabniški vmesnik in skrbijo tako za namestitev podatkovnega modela kot za izvedbo same sinhronizacije med različnimi RDBMS.

3.6 Pantheon

Pantheon Enterprise je PIS za hitro rastoča podjetja, ki stavijo na stabilnost in napredne funkcionalnosti ter potrebujejo prilagodljiv sistem. Širok nabor funkcionalnosti omogoča, da sistem uporabljajo vsi v podjetju, od vodstva do nabave in računovodstva, poleg tega pa lahko izkoristimo tudi vse prednosti poslovne inteligence (nadzorne plošče, analitika, poročila, načrtovanje) [23].

Datalab Pantheon smo uporabili kot referenčni PIS, na katerem smo izvedli integracijo, ki bo služila tudi za ostale PIS.

3.7 Inno Setup

Inno Setup je brezplačni namestitveni čarovnik za operacijski sistem Windows. Prva različica je bila izdana leta 2007, danes pa predstavlja konkurenco mnogim alternativnim plačljivim rešitvam [24].

Je komponenta, ki nam omogoča izvedbo namestitvenega čarovnika v operacijski sistem Windows.

4 Komponente sinhronizatorja

Najprej bomo predstavili vizualni del sinhronizatorja in tako zaradi lažje predstave naredili uvod v podatkovni model, ki ga uporablja sinhronizator.

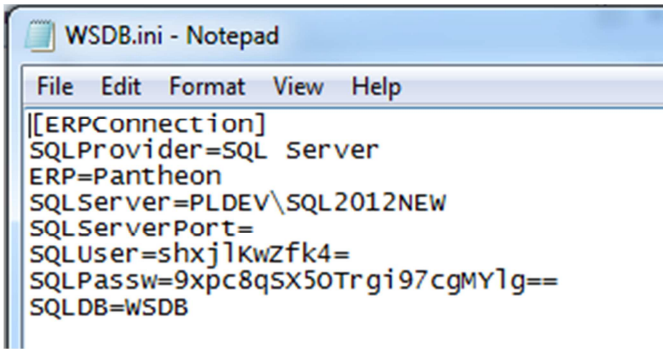
Sinhronizator je sestavljen iz več različnih »glavnih« komponent, ki služijo različnim funkcijam. S tem mislimo predvsem na izvedljive (EXE) datoteke na strani PIS in skripte PHP na strani ST. Namestitveni čarovnik, prav tako ena izmed komponent, poskrbi, da se vse ključne komponente sinhronizatorja namestijo na ciljni računalnik. Deluje podobno kot ostali namestitveni čarovniki za okenske programe. Tako kot mnogi izmed njih je bil izdelan s pomočjo programskega paketa Inno Setup [24]. Ključne komponente, ki so razložene v nadaljevanju, so:

- čarovnik za povezavo na bazo podatkov;
- sinhronizator;
- klient za nastavitve in pregled dnevnika dogodkov;
- servis Windows;
- spletne skripte PHP.

4.1 Čarovnik za povezavo na bazo podatkov

Program služi za kreiranje datoteke INI, ki jo sinhronizator in klient prebereta ob zagonu, in sicer vsebuje kriptirane avtentikacijske podatke za prijavo na bazo podatkov.

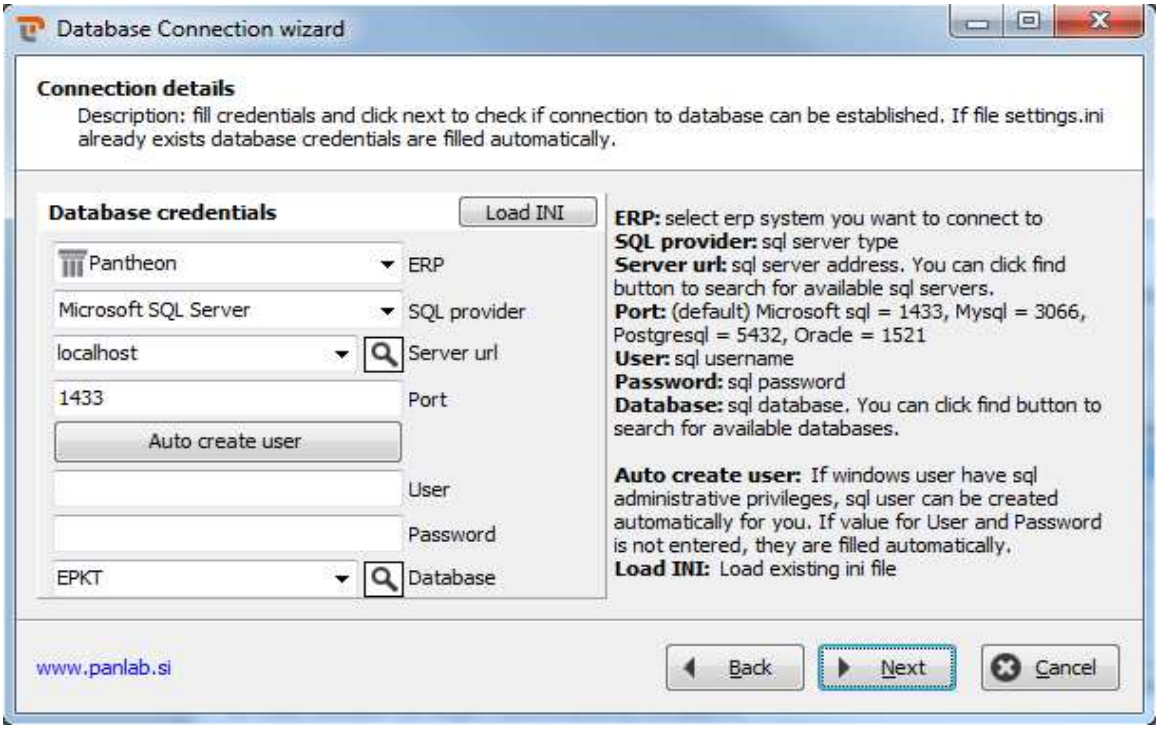
Primer datoteke INI je prikazan na Sliki 4.1.



Slika 4.1: Primer datoteke wsdb.ini.

Kot je prikazano na Sliki 4.1, sta zaradi varnosti tako uporabniško ime kot geslo kriptirana s kriptirnim algoritmom Blowfish [17].

Primer enega izmed korakov v procesu kreiranja povezave na bazo SQL je prikazan na Sliki 4.2.

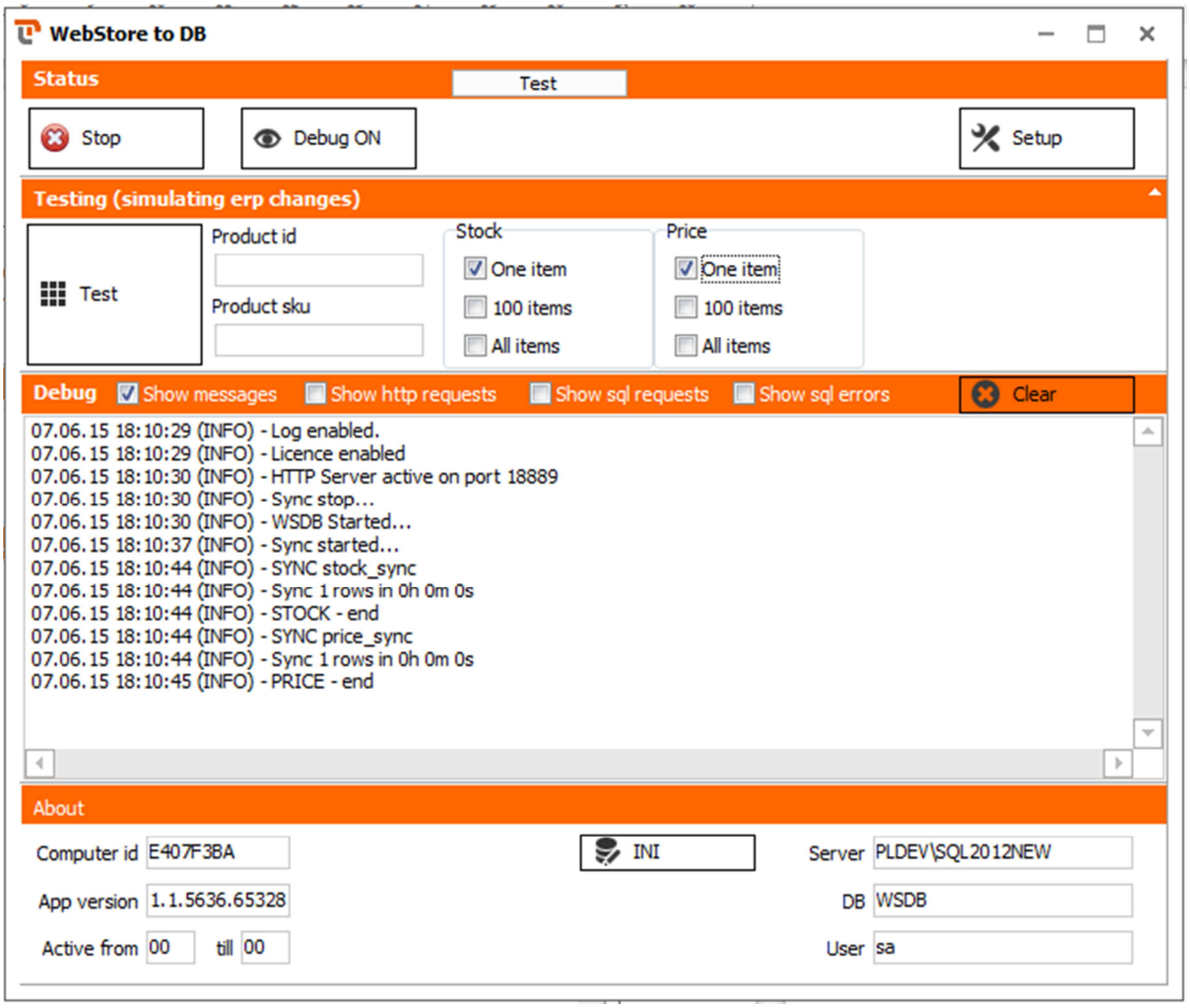


Slika 4.2: Čarovnik za povezavo na bazo podatkov

4.2 Sinhronizator

Sinhronizator vsebuje jedro logike za sinhronizacijo in je ključnega pomena v procesu sinhronizacije. Ker je namenjen avtomatični obdelavi podatkov, ne vsebuje prav veliko

uporabniških opcij, ampak samo tiste, ki so nujno potrebne: gumb za zagon in ustavitev sinhronizacije, bližnjico za zagon klienta (Setup) in avtentikacije (INI), možnosti za testiranje (podrobneje opisano v poglavju Testiranje) in opcije za izbiro prikaza procesnih informacij. V primeru načina »debug« se evidentira praktično vsak korak, ki se zgodi v procesu sinhronizacije, kar je zelo koristno, ko želimo razhroščevati probleme, vključimo pa lahko tudi vodenje dnevnika dogodkov, ki jih pošiljamo na strežnik SQL (od PIS in od ST).



Slika 4.3: Zaslonska maska sinhronizatorja.

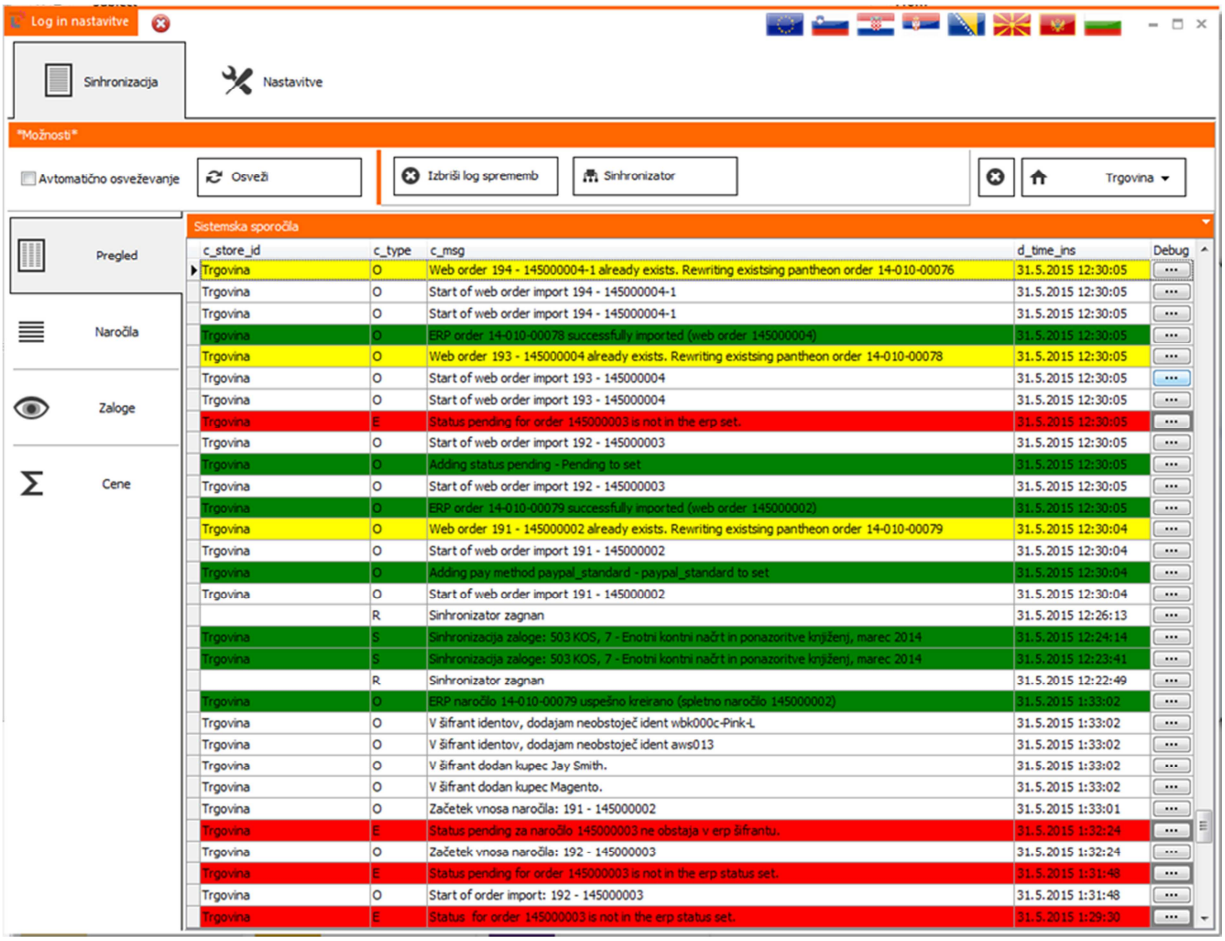
4.3 Zaslonska maska klienta

Klient se uporablja za interakcijo s sinhronizatorjem, upravljanje z nastavitvami in pregledovanje dnevnika dogodkov. Omogoča vpogled v podatke na strani PIS in v podatke na strani ST. Razdeljen je v dve glavni kategoriji, in sicer »sinhronizacija« in »nastavitve«, od

katerih vsebuje vsaka podkategorije, ki predstavljajo posamični modul oz. funkcionalnost sinhronizatorja.

4.3.1 Pregled dnevnika dogodkov

Omogoča vpogled v dnevnik dogodkov, ki jih izvaja sinhronizator, obarvan glede na tip sporočila (plt_message_settings.c_severity). V primeru napake se v stolpec »debug« zapišejo koraki od začetka iteracije do napake (sporočila, ki se sicer prikažejo v sinhronizatorju v načinu »debug«).

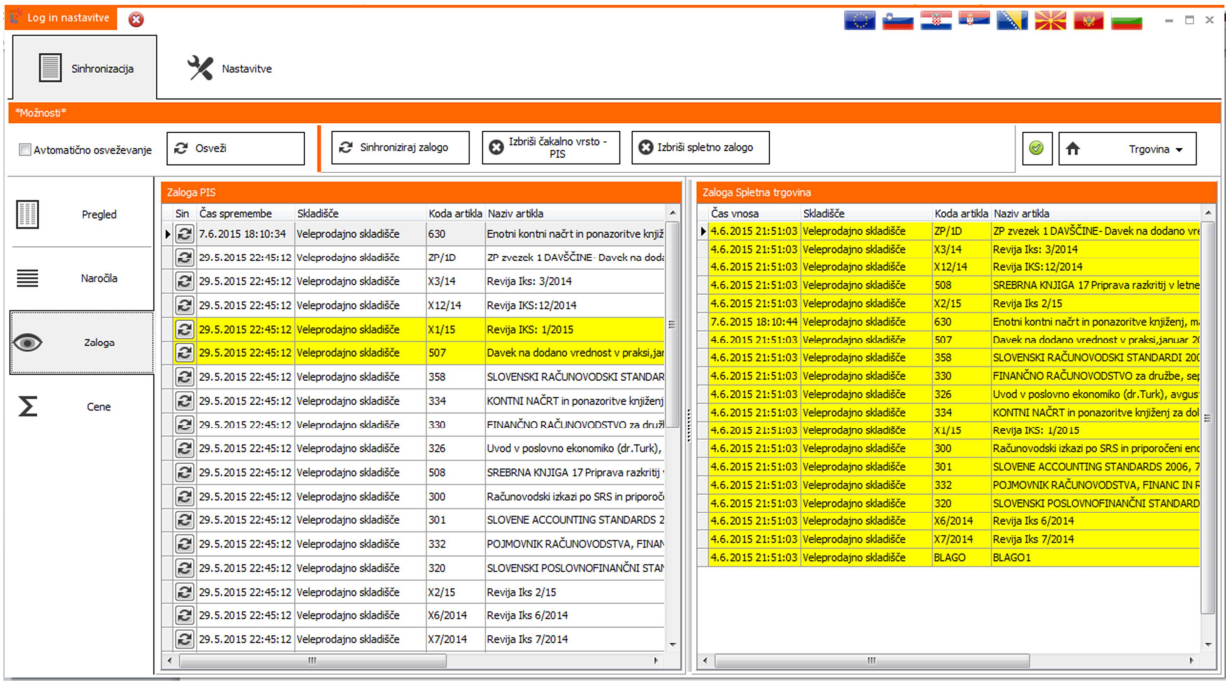


Slika 4.4: Zaslonska maska dnevnika dogodkov.

4.3.2 Pregled sinhronizacije zaloge

Na Sliki 4.5 je prikazana celotna zaloga na strani PIS. Na desni strani je celotna zaloga na strani ST. Zaloga v PIS se prebere iz standardiziranega pogleda plv_stock, v trgovini pa iz tabele plt_stock. V obeh primerih je struktura podatkov enaka za vse PIS in za vse ST, ki kot »backend« uporabljajo tabele SQL. Z rumeno barvo so označeni vsi tisti zapisi, ki so se

spremenili in še niso preneseni ali v ST ali iz prehodnih tabel v uradne strukture ST. Podoben princip se uporablja tudi pri ostalih dveh modulih.



Slika 4.5: Zaslonska maska sinhronizacije zaloge.

4.4 Servis Windows

Predstavlja nevizualno komponento, ki se namesti kot »servis Windows« in skrbi za zagon in ustavljanje sinhronizatorja. Sinhronizator mora biti v produkcijskih okoljih zagnan neprekinjeno, saj lahko nepredvideno nedelovanje pomeni probleme za uporabnike (če se npr. zaloga ne osveži pravočasno, se lahko zgodi, da spletni kupci kupujejo iz neobstoječe zaloge, ki je bila npr. predhodno prodana kupcem v fizični poslovalnici). Zaradi tega razloga je bil kreiran dodaten program (Watchdog), ki se aktivira ob zagonu operacijskega sistema (če je tako nastavljeno) in vsakih nekaj sekund preveri, ali je sinhronizator aktiven oz. ali je sinhronizator zaradi neznane napake obstal. V tem primeru ga prisilno ustavi in na novo zažene.

5 Nivo sinhronizacije SQL

5.1 Uvod v podatkovni model

Postavitev in namestitev podatkovnega modela sinhronizacije predstavlja enega izmed najzahtevnejših delov sinhronizacije (obsega približno 5.000 vrstic programske kode). Generalno je sestavljen iz dveh delov: odvisnega in neodvisnega od informacijskega sistema. Predstavili bomo samo neodvisen del, saj le-ta tvori jedro sinhronizatorja. Lahko rečemo, da večina logike sinhronizacije temelji predvsem na podatkovnem modelu. Napake v njegovi zasnovi se odražajo v funkcionalnih napakah pri delovanju. Posledično to tudi pomeni, da so lahko funkcionalne napake iz naslova nepravilnega podatkovnega modela kasneje težko popravljive, zato je temu področju treba nameniti dovolj časa (težko je spreminjati temelje). Drugače povedano: pred izgradnjo podatkovnega modela moramo dobro razumeti, kaj želimo doseči kot končni produkt, kakšne funkcionalnosti bo produkt vseboval in kako jih želimo realizirati.

Kot uvod v podatkovni model smo opisali funkcionalne potrebe sinhronizacije iz uporabniškega oz. bolj splošnega vidika, in sicer swmo se omejili na sinhronizacijo cen, zaloge in spletnih naročil.

Sinhronizacija zaloge je verjetno ena izmed najpomembnejših funkcionalnosti sinhronizacije s ST na splošno. Končni kupci se ne vračajo po nakupe v tiste ST, ki nepravilno prikazujejo stanje zaloge blaga. Ko kupec izvede naročilo, pričakuje blago v nekaj dneh. Če se naročilo večkrat zapored ne realizira pravočasno zaradi pomakanja zaloge, bo skoraj zagotovo poiskal alternativno trgovino, če le-ta obstaja. Poleg tega je prodajalec zakonsko obvezen dostaviti blago v roku 3 delovnih dni po prejemu plačila, drugače mora kupcu plačati zakonsko določene zamudne obresti, pravi 41. člen Zakona o varstvu potrošnikov [29].

Primarno se informacija o stanju zaloge nahaja v PIS in se spreminja glede na prevzeme, izdajo in rezervacije blaga. V podjetjih, kjer se stanje zaloge ne spreminja pogosto, bi proces usklajevanja zaloge ST z zalogo v PIS lahko izvajali ročno, kar pa je skoraj nemogoče že pri malo večjem obsegu prodaje. Ravno zato se ponudniki spletnih storitev najpogosteje odločajo ravno za sinhronizacijo zaloge. Sprememba stanja zaloge v PIS se lahko zgodi v različnih skladiščih, različnih poslovalnicah in kadar koli v dnevu. Sinhronizator je tisti, ki mora to

spremembo, glede na nastavitve, zaznati in izvesti posodobitev v ST. ST bi lahko najlažje opisali kot ločeno poslovalnico, kjer je vse avtomatizirano in ne potrebuje zaposlenega, da bi prevzemal naročila.

Sinhronizacija cen večinoma ni tako ključnega pomena kot sinhronizacija zaloge. Tudi ni potrebe po realnočasovnih posodabljanjih v ST, vseeno pa bi se nam delo brez te funkcionalnosti podvajalo. Artiklom v PIS določimo prodajno kalkulacijo, se pravi, pod kakšnimi cenovnimi pogoji jih želimo prodajati. Prodajne cene se lahko določijo na več različnih načinov: a) kot končne, kar pomeni, da so cene statične in se avtomatično ne spreminjajo; b) odvisne od nabavne cene, kar pomeni, da se vpiše delež pribitka na nabavno ceno, ki se lahko spreminja ob vsakem nakupu blaga; c) ostalo. Ne glede na izbrano prodajno politiko mora sinhronizator zaznati spremembo prodajne cene in jo pravilno posodobiti v ST.

Sinhronizacija naročil poteka v smeri ST proti PIS. Funkcionalnost je iz časovnega vidika izredno pomembna, saj nam močno olajša proces zajema naročil kupcev B2B in B2C.

V primeru B2B so kupci večinoma znani vnaprej in z njimi poslujemo že dlje časa. ST jim predstavlja avtomatiziran način komunikacije z dobaviteljem in večino tistega, za kar se je prej uporabljal telefon, lahko poslovni partner opravi sam prek ST. Če si samo predstavljamo telefonski pogovor med dobaviteljem in kupcem, kjer kupca zanimajo tako zaloge kot cene potencialno zanimivih artiklov, in k temu dodamo še čas vnosa kreiranja naročila, je za celoten proces potrebnih vsaj 5−10 minut (podatek je nazoren brez kakršne koli empirične podlage), se pravi lahko en zaposleni v eni uri obdela največ 10 naročil, ob večjih naročenih količinah pa niti to ne. Na drugi strani je ST sposobna sprejeti veliko več kot 10 naročil na uro in tudi kupec lahko na tak način dobi več informacij.

Prodaja B2C s pomočjo ST omogoča večji doseg in vse tisto. Kar je bilo prej mogoče samo prek poslovalnice, ki je bila locirana v točno določenem kraju, z uvedbo ST omogočimo praktično celemu svetu.

Ko kupec izvede naročilo v ST, se mora le-to prenesti v PIS z vsemi pripadajočimi podatki. To pomeni, da se v procesu sinhronizacije v PIS doda novo naročilo. Ker pa naročilo vsebuje podatke, ki so tudi po referenčni integriteti vezani na šifrante, je treba pred vnosom naročila v šifrante dodati kupca in artikle, če še ne obstajajo. Naročilo vsebuje tudi podatke nekaterih drugih šifrantov, ki so vsebovani tako v ST kot v PIS, vendar zapisi niso nujno šifrirani z enakimi kodami. Zaradi tega je treba narediti preslikovalne tabele za način plačila, način dostave in status naročila.

5.2 Splošno o podatkovnem modelu

Objekti SQL so zaradi boljšega nadzora in lažje aplikacije avtorizacij poimenovani po enakem principu. Vsa imena so pisana z malimi črkami (baze »case sensitive«), vsebinski sklopi pa so ločeni s podčrtajem (_).

Vsi objekti SQL se zaradi preglednosti in lažje nastavitve avtorizacij začnejo s predpono »**pl**« in nadaljujejo glede na tip objekta:

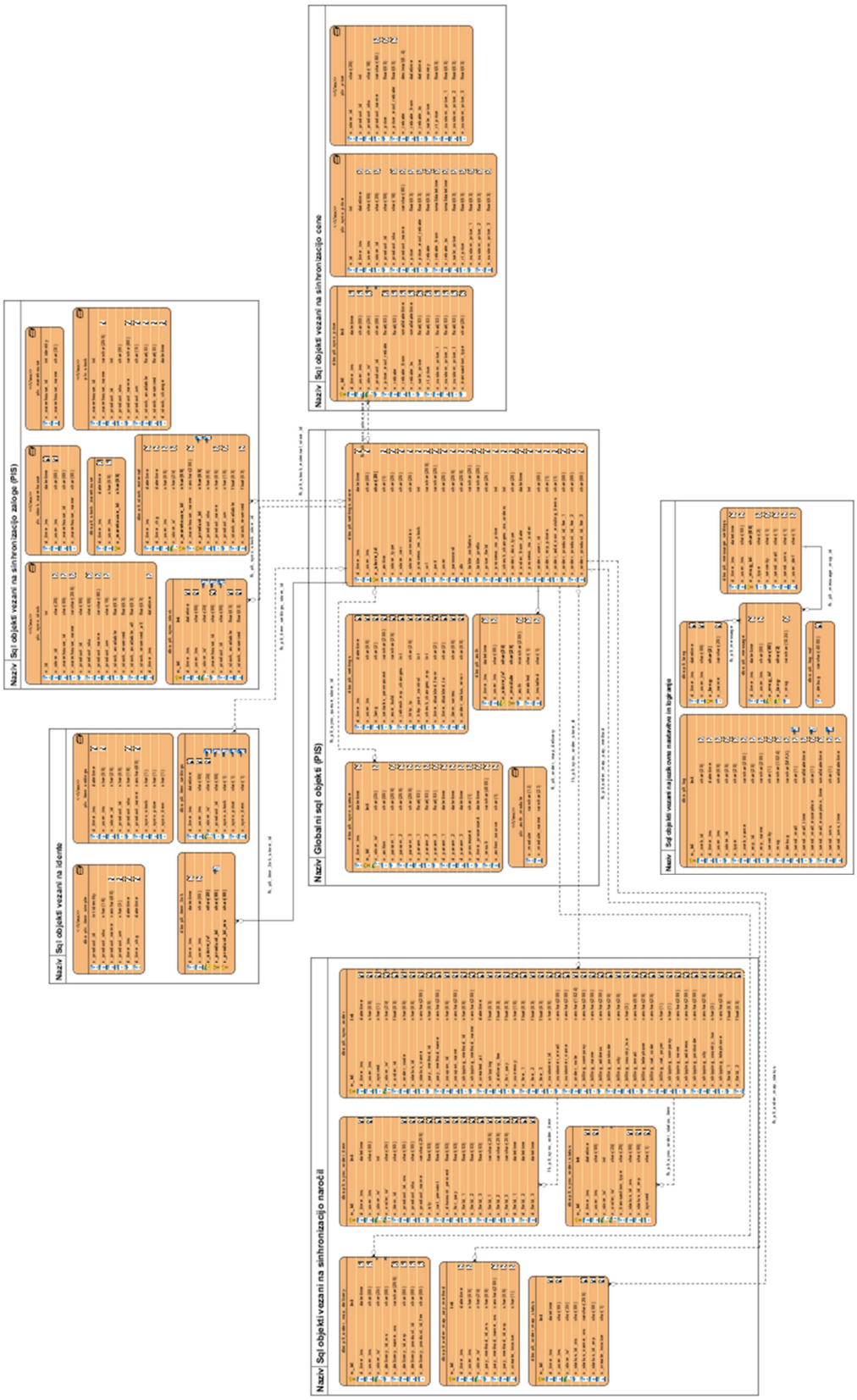
- tabele: **plt**_xxx (primer: plt_settings);
- pogledi: (view): **plv**_xxx (primer: plv_stock);
- procedure: **plp**_xxx (primer: plp_prepare_queue);
- funkcije: **plf**_xxx (primer: plf_message_get);
- prožilci (triggerji): **pltr**_xxx (primer: pltr_auth_store).

Vsa imena stolpcev se začnejo z označbo tipa polja:

- c_: tekstovni tip polja (primer: c_store_id char(25));
- n_: numerični tip polja (primer: n_id int);
- d_: datumski tip polja (primer: d_time datetime).

5.3 Shema ER na strani PIS

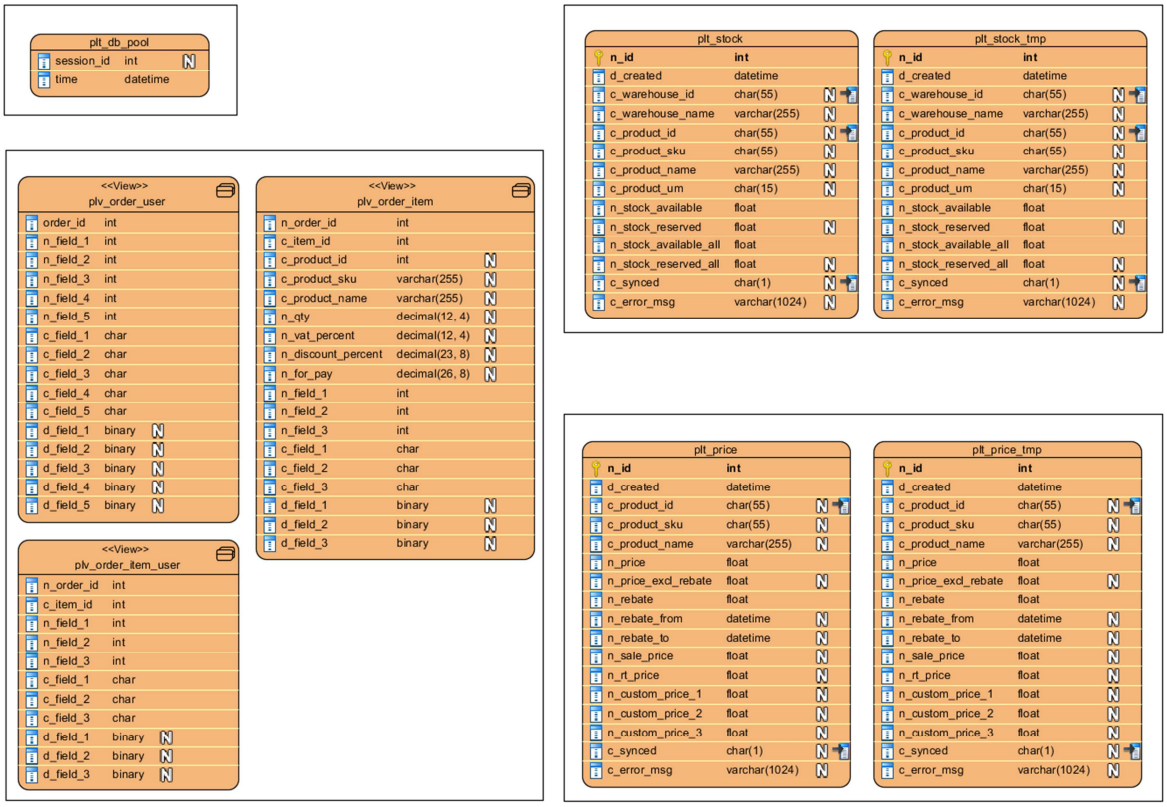
Spodaj je zaradi občutka o podatkovnem modelu prikazana shema tabel, ki jih uporablja sinhronizator na stran PIS. Razlaga sheme se nahaja v poglavjih v nadaljevanju.



Slika 5.1: Shema ER na strani PIS.

5.4 Shema ER na strani ST

Spodaj so prikazani tabele in pogledi, ki se uporabljajo na strani ST. Razlaga sheme se nahaja v poglavjih v nadaljevanju.

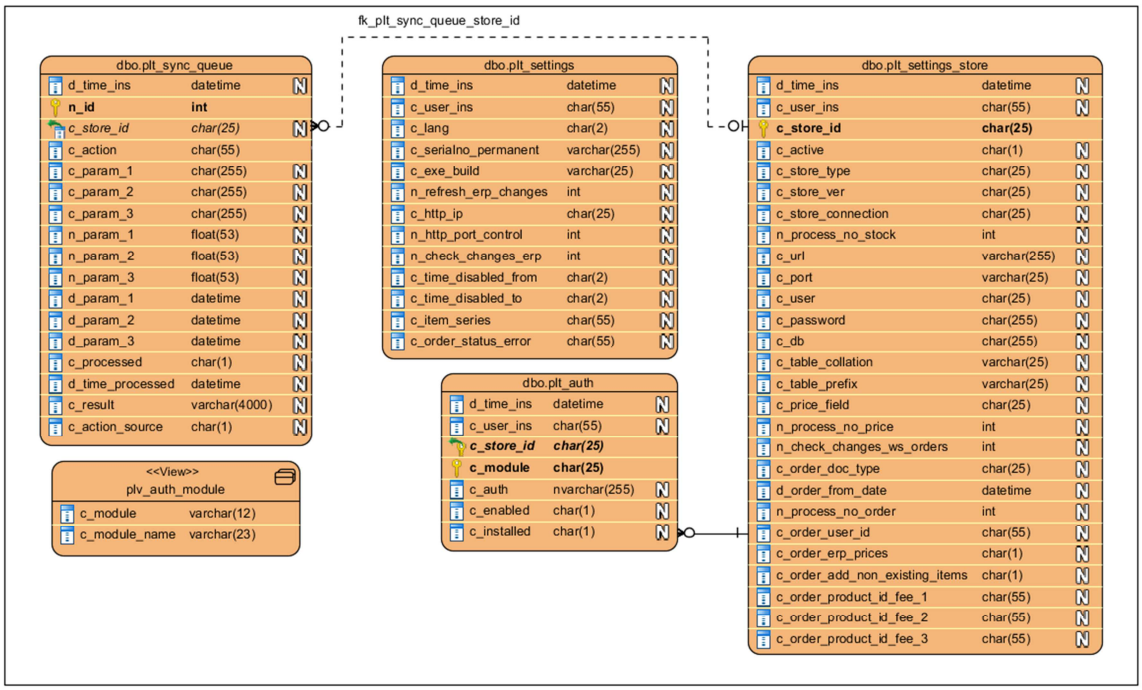


Slika 5.2: Shema ER na strani ST.

5.5 Globalne tabele v PIS

Podatkovni model je z vidika nastavitev sestavljen iz treh nivojev, in sicer iz tabel, ki se vežejo na splošne nastavitve, nastavitve posamezne trgovine in nastavitve posamezne funkcionalnosti znotraj trgovine.

Z globalnimi tabelami smo označili splošne nastavitve in nastavitve, ki se vežejo na posamezno trgovino.



Slika 5.3: Shema ER »globalnih« tabel SQL.

plt_settings – tabela služi splošnim nastavitvam sinhronizatorja. Tukaj se nahajajo nastavitve, kot so naslov in »port« strežnika HTTP (sinhronizator ima vgrajen strežnik HTTP, prek katerega lahko sprejema ukaze), frekvenca spremljanja sprememb v PIS, jezik v uporabi, čas delovanja ipd.

plt_settings_store – podprta je sočasna sinhronizacija iz več različnih podatkovnih virov, se pravi iz več različnih trgovin. Tabela je namenjena nastavitvam, ki se vežejo na posamezno trgovino, npr. avtentikacijske podatke za dostop do trgovine, tip povezave (SQL, web service), nekatere (enodimezionalne) nastavitve sinhronizacijskih modulov ipd.

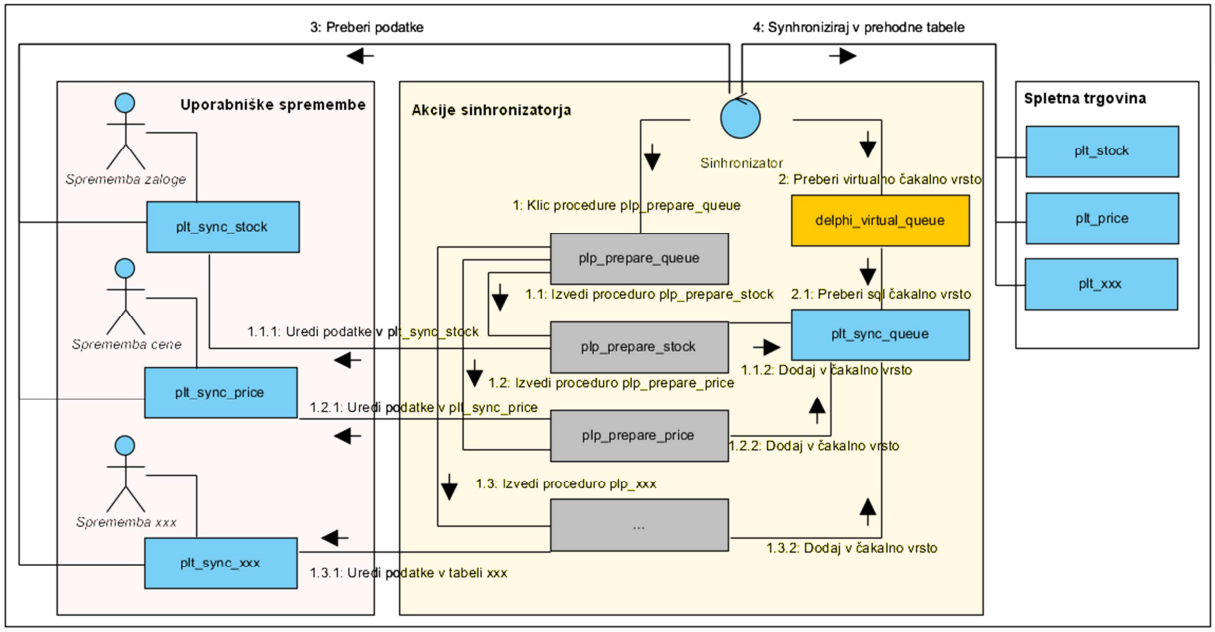
plt_sync_queue – čakalna vrsta, prek katere se izvaja celotna sinhronizacija iz PIS v ST.

5.6 Čakalna vrsta

Sinhronizator v svojem jedru oz. za namene svoje glavne funkcionalnosti ni pisan v večnitnem (multithreading) načinu, saj je pomemben vrstni red izvajanja sinhronizacije, ker so posamezni moduli med seboj odvisni. Zaradi tega se uporablja čakalno vrsto tako v okolju SQL kot v okolju Delphi. V obeh primerih je funkcionalnost čakalne vrste enaka. Prav tako so enake podatkovne strukture, različen je le izvor vhodnih podatkov.

5.6.1 Nivo SQL

Akcije za sinhronizacijo iz PIS v ST se najprej dodajo v tabelo plt_sync_queue. Sinhronizator ob določeni iteraciji (frekvenca je določena v plt_settings.n_check_changes_erp) preveri, ali se v čakalni vrsti nahaja kakšen zapis, ki ima oznako c_processed = 'F'. Če obstaja, to akcijo izvede. Podatki iz tabele plt_sync_queue se interno prenesejo v virtualno tabelo (Delphi), ki je strukturirana enako kot tabela SQL, ki je dejansko osnova za procesiranje sinhronizatorja. Virtualno tabelo je mogoče urejati tudi prek vgrajenega strežnika HTTP, kar je prikazano v naslednjem poglavju. Logika delovanja na nivoju SQL je prikazana na Sliki 5.4.



Slika 5.4: Logika SQL prenosa podatkov iz PIS v ST.

Če v grobem povzamemo Sliko 5.4, je časovno sosledje dogodkov sledeče:

1) uporabnik kreira spremembo;

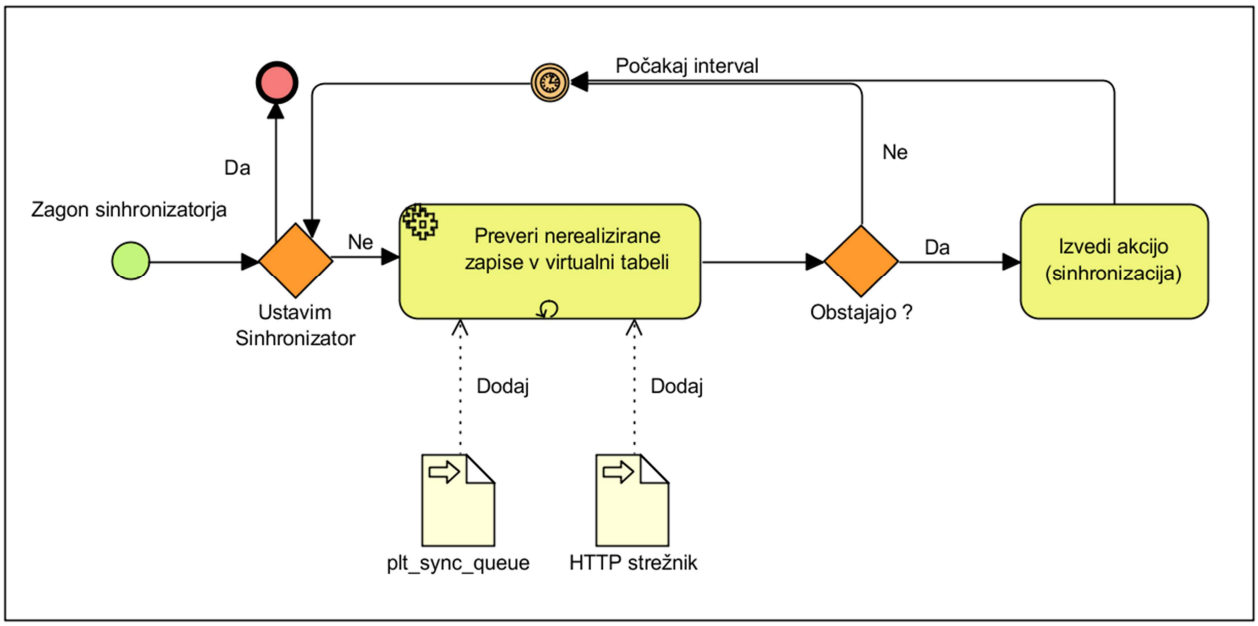
2) sinhronizator kliče krovno proceduro plp_prepare_queue, ki kliče procedure plp_prepare_??? za vsako funkcionalnost posebej. Če obstajajo podatki za sinhronizacijo, se zapišejo v tabelo plt_sync_queue;

3) sinhronizator prebere podatke iz tabele SQL v virtualno tabelo, kar je osnova za izvedbo dejanske sinhronizacije.

Podrobna razlaga Slike 5.4 po časovnih alinejah bi bila:

1. klic procedure plp_prepare_queue, ki interno kliče ostale procedure za pripravo podatkov;
   1.1. izvedi proceduro plp_prepare_stock, katere funkcija je, da:
      1.1.1. uredi zapise v tabeli plt_sync_stock: urejanje med drugim vključuje izbris nepotrebnih zapisov (npr. za idente, kjer ni nastavljeno, da se sinhronizirajo), odstranjevanje duplikatov, določanje, v katero trgovino se podatki sinhronizirajo, itd.;
      1.1.2. vpiše zahtevo za sinhronizacijo v tabelo plt_sync_queue: če po urejanju obstajajo primerni podatki za sinhronizacijo, se grupirano po trgovini dodajo zahtevki za sinhronizacijo v čakalno vrsto, ki se kasneje prebere v virtualno čakalno vrsto znotraj Delphija;
   1.2. izvedi proceduro plp_prepare_price, funkcionalna ideja je enaka kot pri plt_prepare_stock:
      1.2.1. uredi tabelo plt_sync_price;
      1.2.2. dodaj v čakalno vrsto;
   1.3. zgornji princip se lahko aplicira v večini primerov, ko želimo sinhronizirati podatke iz PIS v ST oz. obratno;
2. sinhronizator prebere virtualno čakalno vrsto;
   2.1. v virtualno čakalno vrsto se preberejo zahtevki iz čakalne vrste SQL, ki jo je predhodno napolnila procedura plp_prepare_queue;
3. glede na čakalno vrsto se preberejo podatki v tabelah plt_sync_xxx;
4. prebrani podatki se prenesejo v ST v prehodne tabele: za vsako iteracijo plt_settings.n_check_changes_erp (iteracije se izvajajo vsako sekundo) se zažene procedura plp_prepare_queue, ki interno kliče procedure »prepare« za vse aktivne module (plp_prepare_price, plp_prepare_stock ipd.). Procedura uredi podatke v tabelah »sync« (plt_sync_price, plt_sync_stock) na tak način, da so podatki pripravljeni za sinhronizacijo. V primeru, da obstajajo podatki v tabeli »sync«, se v čakalno vrsto (plt_sync_queue) doda ustrezna zahteva za sinhronizacijo na nivoju ST. Procedure »prepare« so potrebne predvsem zaradi optimizacije hitrosti delovanja PIS, saj v sprožilce ne moremo vgraditi kompleksne logike, zato je le-ta prestavljena na čas dogodka sinhronizacije.

5.6.2 Nivo Delphi



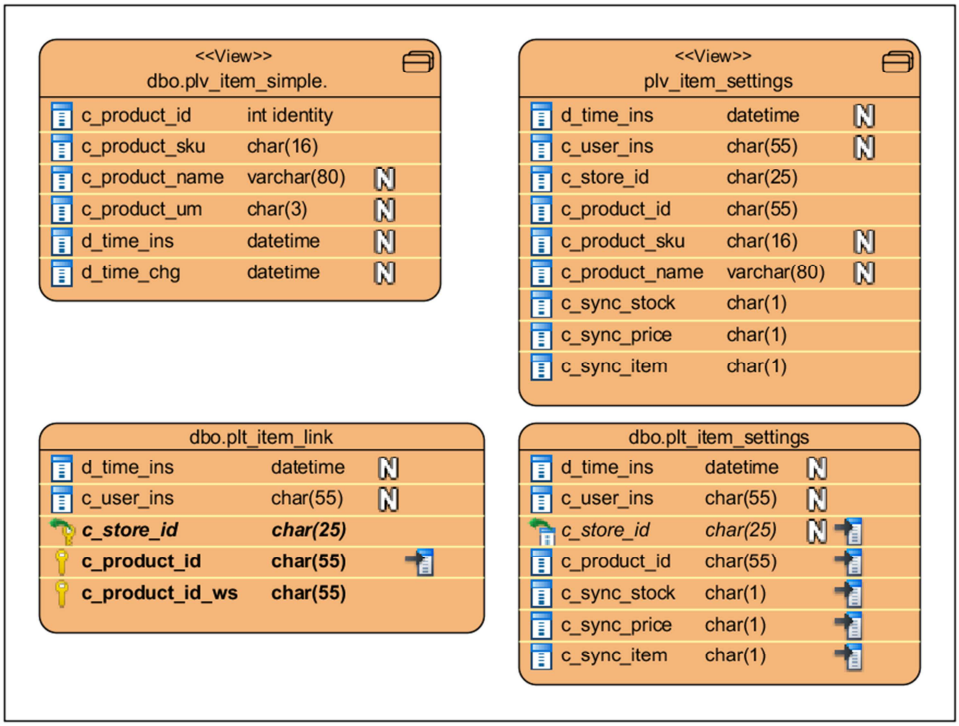
Slika 5.5: Logika procesa sinhronizacije Delphi.

Sinhronizator deluje iterativno, dokler se ne ustavi. V vsaki iteraciji se preveri, ali je treba procesirati virtualno tabelo, v katero se dodaja tako prek baze SQL kot prek strežnika HTTP. Virtualna tabela predstavlja realizacijo tabele SQL v okolju Delphi, ki obstaja samo v računalniškem delovnem spominu (RAM). Poizvedovanje/dodajanje akcij prek strežnika HTTP se dogaja ob uporabniških zahtevah, medtem ko se prek tabele SQL dodaja večinoma z naslova avtomatike, kot je prikazano na Sliki 5.4.

5.7 Nastavitve artiklov

Artikli (identi) so skupna točka vsem sinhronizacijskim modulom.

5.7.1.1 Diagram ER artiklov na strani PIS



Slika 5.6: Shema ER tabel za artikle.

plv_item_simple – »preprost« pogled na artikle PIS. Pogled vsebuje osnovne lastnosti artiklov, kot so id, koda, naziv, enota, mere in čas vnosa/spremembe. Pogled se uporablja v različnih procesih sinhronizatorja.

Kot zanimivost lahko omenimo, da je zaradi nekompatibilnosti med različnimi podatkovnimi strukturami v različnih PIS težko določiti standardiziran pogled na idente PIS. Zaradi tega sta se kreirala dva pogleda, in sicer plv_item_simple (prikazan na Sliki 5.6) in plv_item. plv_item je strukturiran po logiki EAV (entity–attribute–value), kar pomeni, da omogoča dinamično dodajanje poljubnega števila lastnosti, kar se uporablja pri sinhronizaciji identov, vendar je to izven tematike diplomskega dela.

plv_item_settings – v tej tabeli se nahajajo nastavitve, katere funkcionalnosti se sinhronizirajo za posamezni ident. Tako lahko npr. za določen ident vključimo posodabljanje spletne zaloge, medtem ko izključimo posodabljanje njegove cene.

plv_item_settings – pogled na plt_item_settings z nekaj dodanimi podatki.

plt_item_link – povezovalna oz. mapirna tabela med kodo spletnega identa in kodo identa PIS.

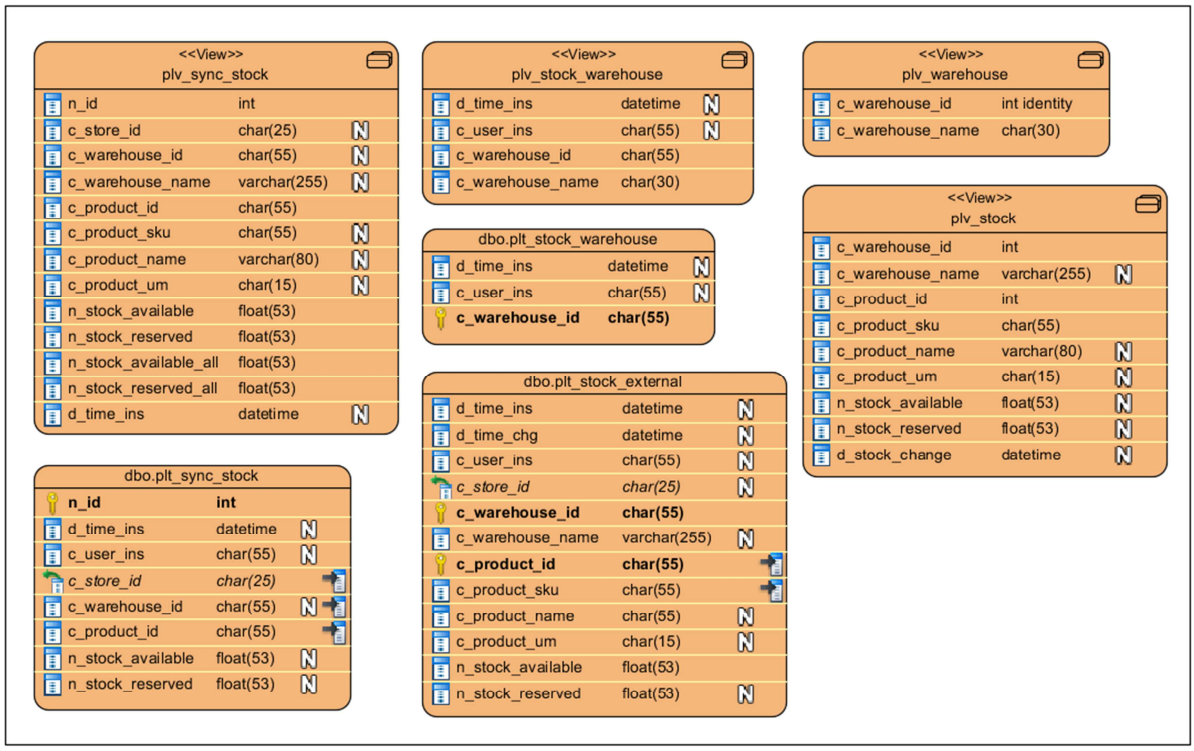
5.8 Sinhronizacija zaloge

Sinhronizacija zaloge je modul, ki skrbi za prenos zaloge iz PIS v ST. V nastavitvah lahko določimo, katera skladišča v PIS vplivajo na spletno zalogo, posodobitev zaloge pa se lahko dogaja skoraj realnočasovno. Poleg tega je možno določiti za vsak ident posebej, ali se zanj zaloga sinhronizira ali ne (privzeto se sinhronizira za vse).

Ker se za pridobivanje informacij o zalogi uporablja pogled (view), je možno razširiti delovanje sinhronizatorja, da se pri posodobitvah zaloge v ST lahko upošteva tudi zaloga iz različnih »zunanjih« podatkovnih virov, npr.: drugih baz SQL (npr. skupna zaloga več različnih podjetij), oz. se lahko upošteva zunanja zaloga dobavitelja, ki je v PIS ne vodimo (se jo npr. uvaža prek datoteke XML). To omogoča pravilno posodabljanje spletne zaloge tudi v primeru nadaljnje prodaje.

Spletnim identom, ki v PIS ne obstajajo, se zaloga ne spreminja.

5.8.1 Diagram ER, vezan na sinhronizacijo zaloge na strani PIS



Slika 5.7: Shema ER za sinhronizacijo zaloge na strani PIS.

plt_sync_stock – tabela hrani idente, za katere je treba izvesti sinhronizacijo. Polni se praviloma prek prožilcev, ko se zaloga v PIS spremeni.

plv_sync_stock – omogoča pogled na zgornjo tabelo, s tem da so dodane določene opisne informacije (npr. naziv skladišča, naziv identa ipd.).

plt_stock_external – tabela se uporablja za sinhronizacijo zunanje zaloge, se pravi zaloge, ki ni del PIS.

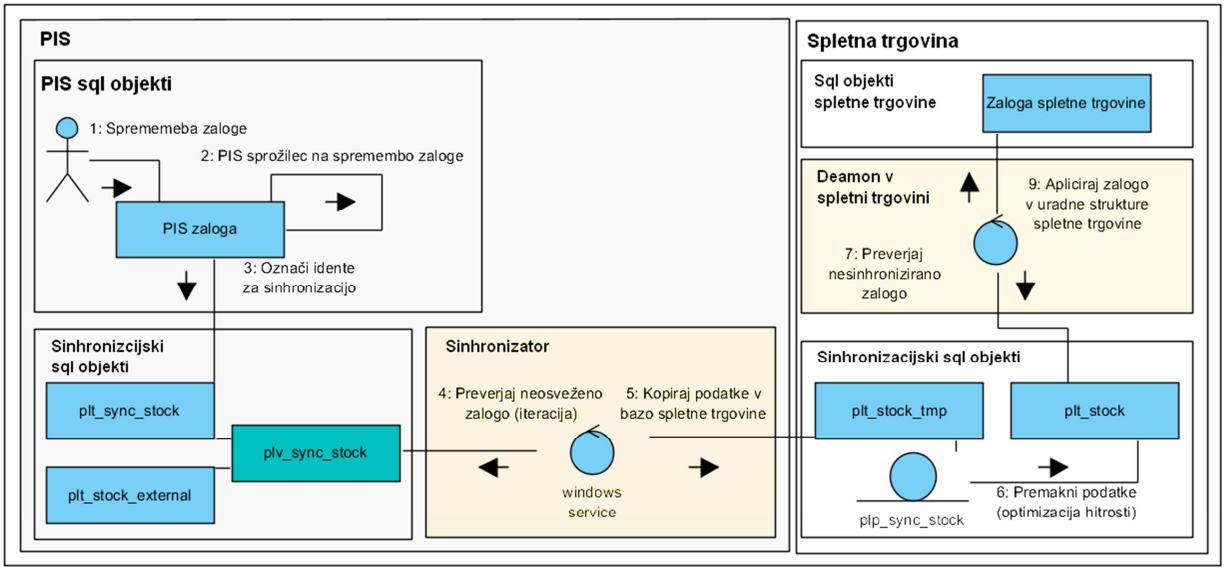
plt_stock_warehouse – tabela vsebuje listo skladišč, ki vplivajo na spletno zalogo.

plv_stock_warehouse – pogled, ki poleg zgornjih podatkov prikazuje še naziv skladišča.

plv_warehouse – pogled, ki vrača skladišča, zavedena v PIS.

plv_stock – pogled, ki vrača zalogo za skladišča, vnesena v tabelo plt_stock_warehouse.

5.8.2 Komunikacijski diagram



Slika 5.8: Komunikacijski diagram sinhronizacije zaloge.

Ko se zgodi sprememba zaloge v PIS, se spremembo prek prožilcev zapiše v tabelo plt_sync_stock. Spremembo zaloge lahko sproži avtomatika ali končni uporabnik. Z vidika

sinhronizatorja to ni pomembno. Podatki potem »čakajo« v tabeli, dokler jih sinhronizator ne zazna in prenese na stran ST (intervali preverjanja so lahko tudi sekundni). Zaradi performančnih razlogov (ni potrebno preverjanje, ali zapis obstaja, saj je tabela plt_stock_tmp vedno prazna) se podatki najprej dodajo v tabelo plt_stock_tmp in od tam naprej prek procedure v tabelo plt_stock.

Logika na strani ST je podobna, in sicer podatki »čakajo«, dokler jih »Daemon« na strani ST ne zazna in aplicira v ST. »Daemon« je lahko pisan v katerem koli programskem jeziku in je tako prilagojen za katero koli ST.

5.8.3 Primer programske kode prožilca na strani PIS (Microsoft SQL)

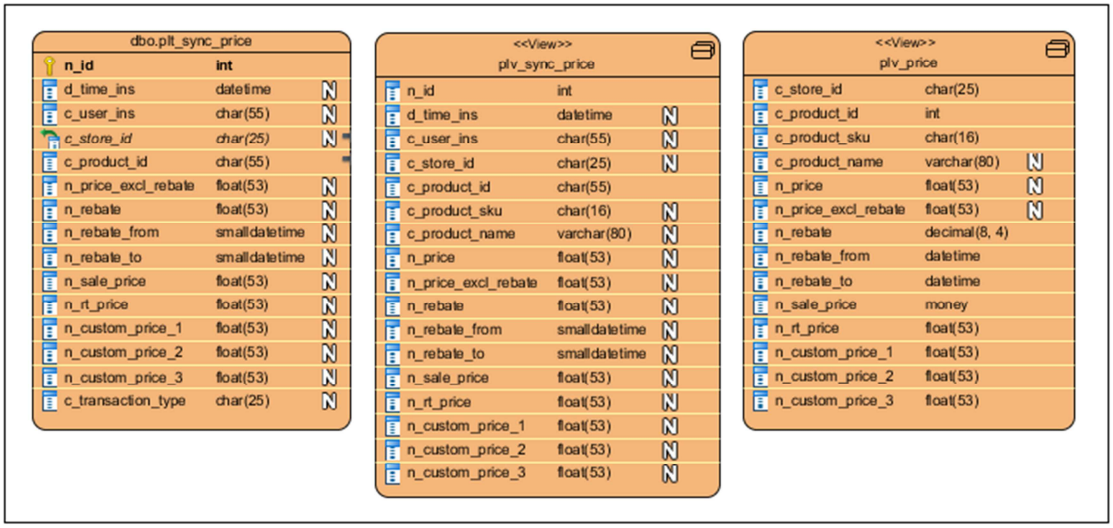
```
IF OBJECT_ID ('dbo.pltr_stock_sync','TR') IS NULL
exec('create trigger dbo.pltr_stock_sync on tHE_Stock WITH ENCRYPTION After insert, update as select 1 ')
GO
alter trigger dbo.pltr_stock_sync
on tHE_Stock
WITH ENCRYPTION
After insert, update
as

insert into dbo.plt_sync_stock(c_store_id, c_warehouse_id, c_product_id, n_stock_available, n_stock_reserved)
select null, S.anQid, SI.anQid, null, null
    from INSERTED I
    left join DELETED D
            on D.acWarehouse = I.acWarehouse
            and D.acIdent = I.acIdent
    join tHE_SetItem SI on SI.acIdent = I.acIdent
    join tHE_SetSubj S on S.acSubject = I.acWareHouse
    join plt_stock_warehouse sw on sw.c_warehouse_id = S.anQid
        where 1 = 1
        and (isnull(D.anStock,0) != I.anStock
            or isnull(D.anReserved,0) != I.anReserved
            )
            group by S.anQid, SI.anQid
```

Slika 5.9: Primer prožilca za evidentiranje sprememb zaloge (MS SQL).

5.9 Sinhronizacija cen

Modul omogoča avtomatično sinhronizacijo cen iz PIS v ST. Sinhronizirajo se lahko različne cene. Določene nastavitve prepustimo končnemu uporabniku, določene nastavitve pa se nastavijo pri namestitvi struktur SQL. Na kakšen način se prebira cene iz PIS, je v celoti logika pogleda plv_price.

5.9.1 Diagram ER, vezan na sinhronizacijo cen na strani PIS



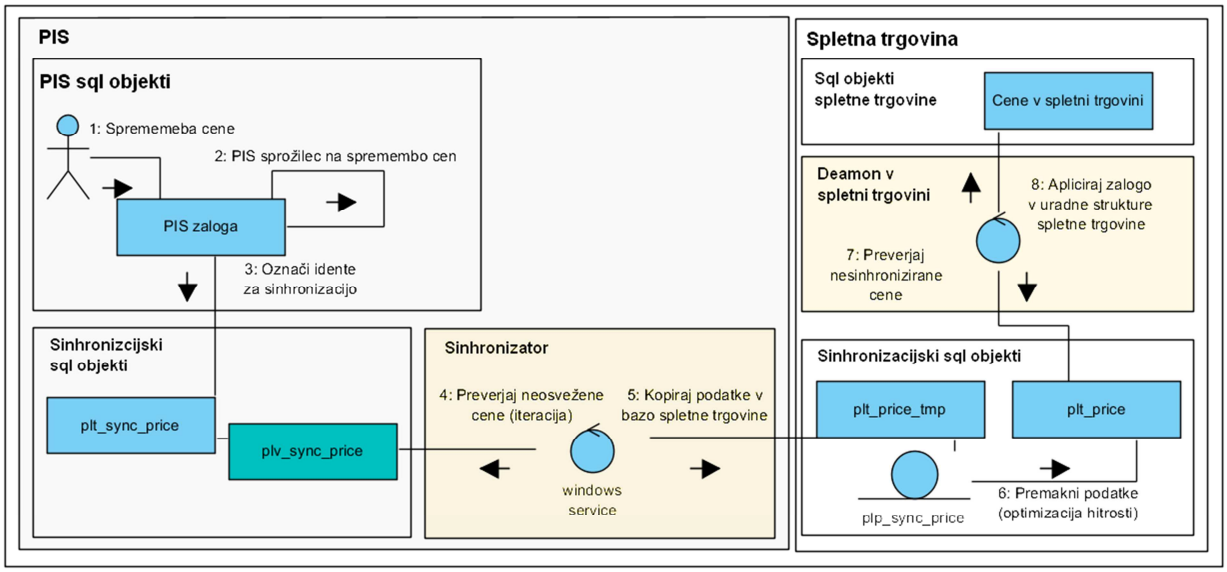
Slika 5.10: Tabele SQL za sinhronizacijo cen na strani PIS.

plt_sync_price – tabela vsebuje idente, za katere je treba izvesti sinhronizacijo cen.

plv_sync_price – pogled na zgornjo tabelo z dodanimi nekaterimi opisnimi informacijami (nazivi).

plv_price – pogled, ki vrača cene za vse idente PIS. Nastavitve, katere cene se prikazujejo (PIS ima lahko več različnih cen), se lahko določi v tabeli plt_settings_store.c_price_field. Poleg tega je mogoče spremeniti tudi sam pogled plv_price.

5.9.2 Komunikacijski diagram



Slika 5.11: Komunikacijski diagram sinhronizacije zaloge.

Proces sinhronizacije cen je zelo podoben procesu sinhronizaciji zaloge. Ko se spremeni cena v PIS, se idente doda v čakalno vrsto za sinhronizacijo. Za razliko od sinhronizacije zaloge se cene lahko vodi v različnih tabelah in stolpcih, zato je logika na strani PIS in v ST bolj kompleksna. Ostali koraki so skoraj identični kot pri sinhronizaciji zaloge.

5.9.3 Primer kode Delphi za posodabljanje v ST

V nadaljevanju je prikazan primer kode Delphi, ki skrbi za posodabljanje podatkov med bazami. Koda je tako generična, da se ta del uporablja za posodabljanje vseh modulov (vseh tabel) in katere koli baze.

Variabla slFields vsebuje vse stolpce, ki so na voljo v določeni poizvedbi SQL. source_dataset vsebuje podatke iz izvorne baze podatkov, dest_dataset pa vsebuje povezavo na ciljne SQL strukture, kamor vpisujemo podatke. Podatke prenesemo iz izvorne v ciljno bazo, tako da na prvem nivoju izvedemo iteracijo prek izvornih zapisov, na drugem nivoju pa iteracijo prek stolpcev, ki jih je treba posodobiti. Podatke v ciljni bazi posodabljamo oz. dodajamo na način »ciljna_tabela.stolpec = izvorna_tabela.stolpec«.

```
function fSetDataSet: Boolean;
var i: Integer;
begin
    i := 0;
    while i < slFields.Count-1 do begin
        // STOLPEC SE NE SINHRONIZIRA -> PRESKOČI
        if SL_SETTINGS.Values['exclude_fields'] = slFields[i] then begin
            i := i + 3;
            continue;
        end;
        // VREDNOST STOLPCA JE NULL -> PRESKOČI
        if source_dataset.FieldByName(slFields[i]).IsNull then begin
            i := i + 3;
            continue;
        end;

        // PRAVILNO POSODOBI GLEDE NA TIP VREDNOSTI
        if slFields[i+1] = 'Integer'
        then dest_dataset.FieldByName(slFields[i]).AsInteger :=
                source_dataset.FieldByName(slFields[i]).AsInteger
        else if slFields[i+1] = 'Float'
        then dest_dataset.FieldByName(slFields[i]).AsFloat :=
                source_dataset.FieldByName(slFields[i]).AsFloat
        else if slFields[i+1] = 'DateTime'
        then dest_dataset.FieldByName(slFields[i]).AsDateTime :=
                source_dataset.FieldByName(slFields[i]).AsDateTime
        else if slFields[i+1] = 'Raw'
        then dest_dataset.FieldByName(slFields[i]).Value :=
                source_dataset.FieldByName(slFields[i]).Value
        else dest_dataset.FieldByName(slFields[i]).AsString :=
                Copy(source_dataset.FieldByName(slFields[i]).AsString,1,StrToInt(slFields[i+2]));

        i := i + 3;
    end;
    Result := True;
end;
```

Slika 5.12: Primer programske kode Delphi za sinhronizacijo med različnimi bazami SQL.

5.10 Sinhronizacija naročil

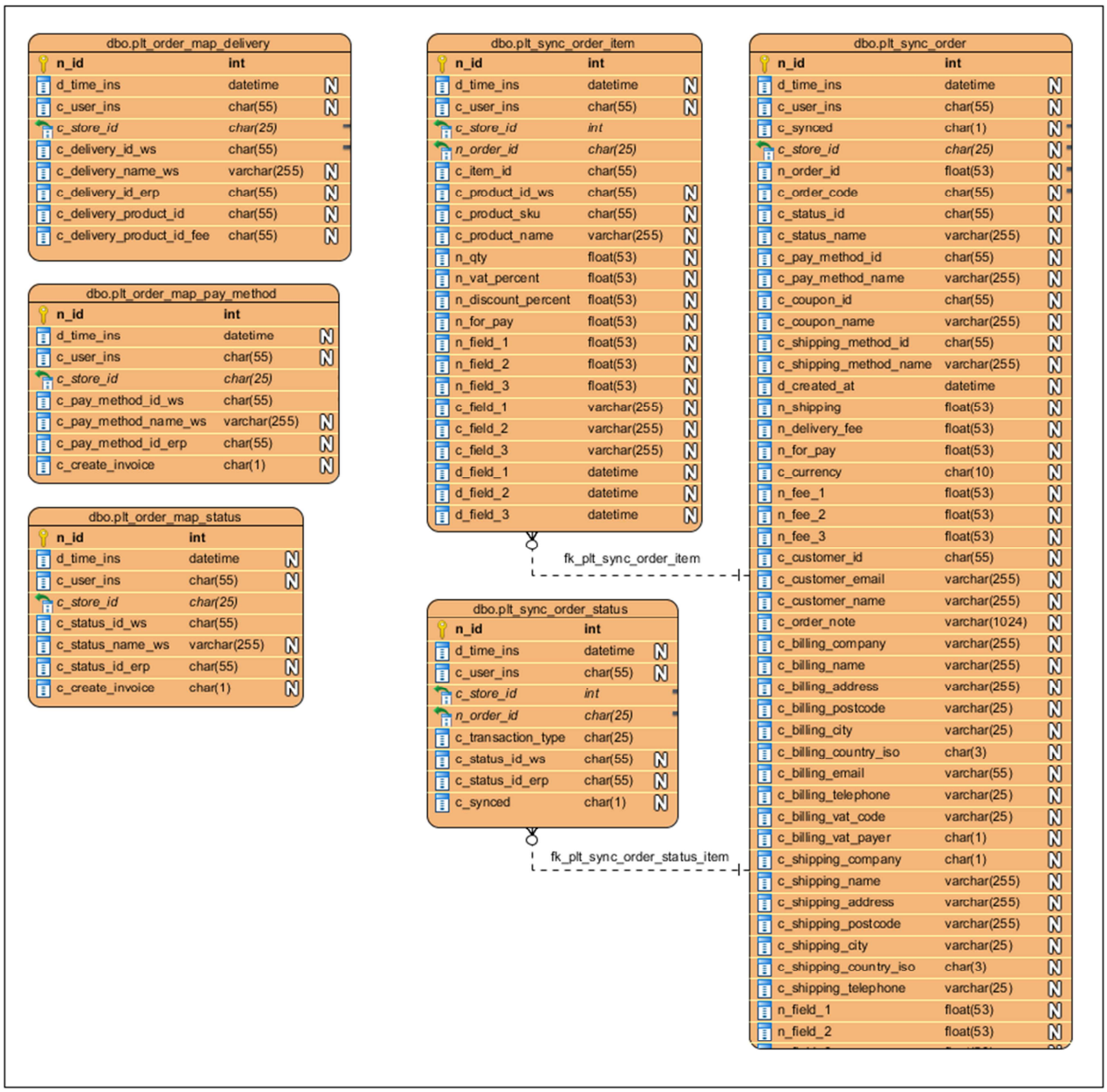
Sinhronizacija naročil se izvaja iz ST v PIS. Omogoča prenos naročil, ki jih kreirajo kupci v ST v naročila PIS. V procesu sinhronizacije se v PIS avtomatično dodajo/kreirajo tako kupci (prejemniki blaga in plačniki) kot tudi identi, če le-ti še ne obstajajo. To omogoča velike prihranke časa, ki bi ga sicer potrebovali, če bi podatke vnašali ročno. Poleg tega se izboljša točnost podatkov, saj se pri ročnih vnosih lahko zgodijo napake. Posledica tega so lahko napačne izdobave in kasnejše reklamacije.

Avtomatičen vnos identov omogoča tudi model nadaljnje prodaje, ko imamo v ST objavljene tudi idente, ki so na zalogi samo pri naših dobaviteljih in jih v PIS niti ne vodimo, dokler se ne zgodi naročilo takega identa.

Sinhronizacija deluje tako, da se periodično preverja nova naročila v ST in jih prenaša v PIS. Nova naročila se preverjajo glede na n_order_id, saj so časi v trgovini velikokrat nastavljeni

napačno oz. se občasno spreminjajo (časovne nastavitve strežnika gmt). V bazi ST se kreirata dva ključna pogleda SQL, in sicer plv_order in plv_order_item, ki vračata vse relevantne podatke za prenos naročila v PIS. Pogleda se nastavita v procesu namestitve za vsako trgovino in/ali verzijo trgovine posebej.

5.10.1 Diagram ER, vezan na sinhronizacijo naročil na strani PIS



Slika 5.13: Tabele SQL za sinhronizacijo naročil PIS.

Na strani ST so skoraj identične strukture, a s to razliko, da so v trgovini pogledi na podatke naročil, v PIS pa tabele, kamor se shranijo podatki iz teh pogledov.

plt_sync_order – kopija PIS spletnih naročil (podatke se pridobi 1 : 1 s spletnega pogleda SQL plv_order v ST).

plt_sync_order_item – pozicije spletnih naročil (podatke se pridobi s spletnega pogleda SQL plv_order_item v ST).

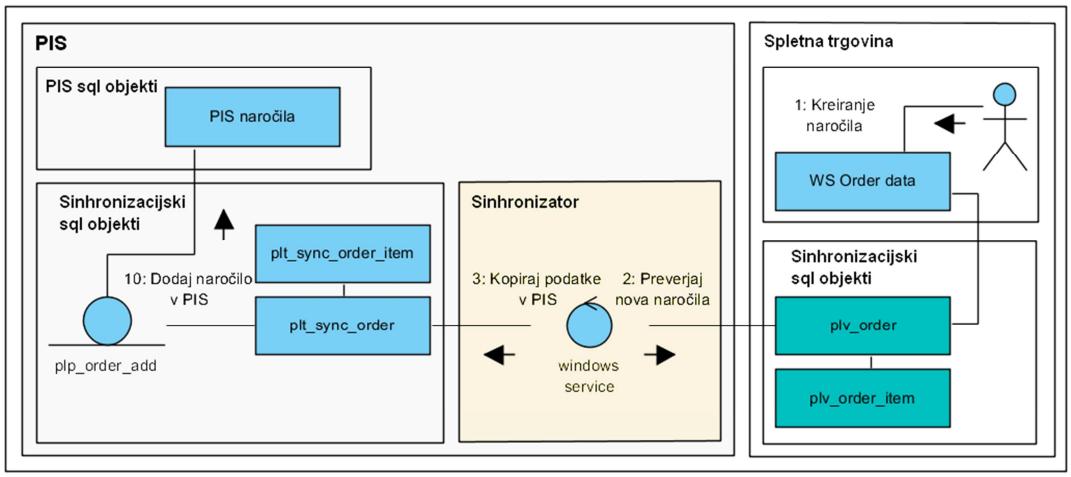
plt_sync_order_status – zgodovina sprememb statusov spletnih naročil. Neformalno pravilo oz. praksa ST je, da se oddano naročilo ne spreminja. V primeru spremembe naročila se starega razveljavi in kreira novo kopijo. To pa seveda ne velja za status naročila, ki se tekom procesa izdobave naročila lahko večkrat spremeni. Zaradi tega se vse spremembe statusov zapisujejo v ločeno tabelo, iz katere je razvidno, kdaj je bil status spremenjen, saj se pogosto na spremembo statusa vežejo tudi druge akcije, kot so npr. pošiljanje SMS-sporočil, avtomatično kreiranje računov, stornacija rezerviranih identov itd.

plt_order_map_delivery – mapiranje načinov dostave med PIS in ST. Kode spletnih načinov dostave se večinoma razlikujejo od kode v PIS, zato je potrebna povezovalna oz. mapirna tabela med obema sistemoma.

plt_order_map_pay_method – povezovanje načinov plačil med PIS in ST.

plt_order_map_status – povezovanje statusov med PIS in ST.

5.10.2 Komunikacijski diagram sinhronizacije naročil



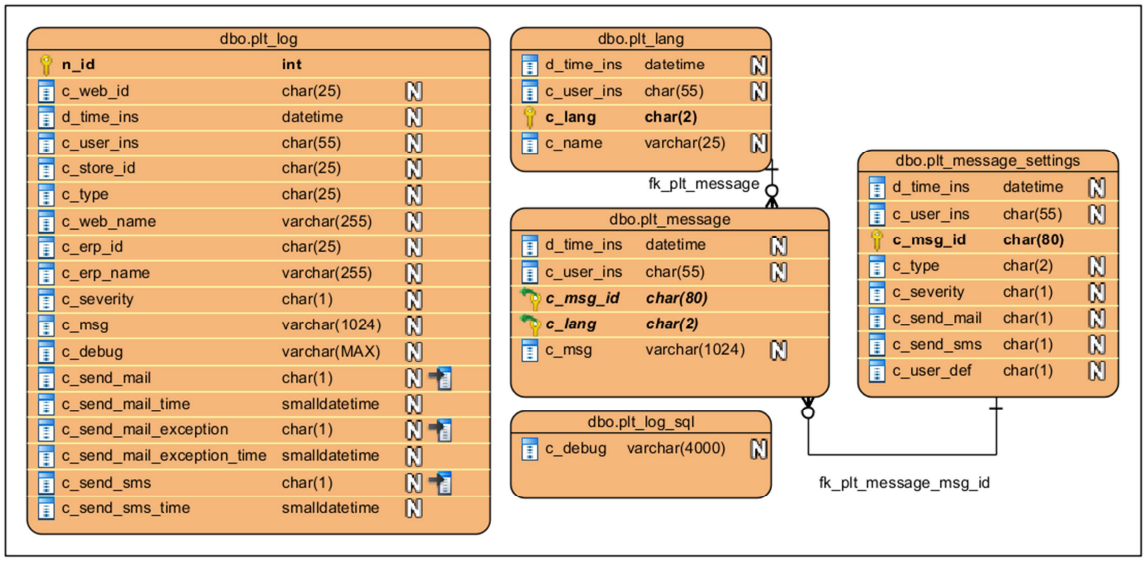
Slika 5.14: Komunikacijski diagram za uvoz spletnih naročil

Sinhronizator na strani PIS periodično preverja, ali obstajajo nova naročila prek standardiziranega spletnega pogleda SQL plv_order. Če zazna novo naročilo, ga prenese v prehodno tabelo na strani PIS in kliče proceduro plp_order_add, ki je posebej prilagojena vsakemu PIS. Dokler naročilo ni zaključeno, se sinhronizirajo tudi status naročila oz. spremembe statusa naročila.

5.11 Vodenje dnevnikov

Če ne želimo, da nas uporabnik kliče ob vsakem najmanjšem problemu, je vodenje dnevnikov v avtomatizirani procesni aplikaciji ključnega pomena. Vsi ključni dogodki v procesu sinhronizacije se zabeležijo, poleg tega pa vsaka napaka vključuje revizijsko sled procesa sinhronizacije (stolpec c_debug). Da ne prihaja do cikličnosti, se vse napake zapišejo v dnevnik in preskočijo, možno pa je obveščanje uporabnika na elektronski naslov. Zaradi internacionalizacije aplikacije je treba zagotoviti večjezično podporo zaslonske maske in seveda tudi večjezično vodenje dnevnikov.

Tabele SQL, ki so vezane na jezikovne nastavitve in vodenje dnevnikov, so prikazane na Sliki 5.15.



Slika 5.15: Shema tabel ER za vodenje dnevnikov in jezikovne nastavitve.

plt_lang – kode in imena podprtih jezikov.

plt_message – vsa sporočila, ki se lahko zabeležijo tekom sinhronizacije, so zapisana v tej tabeli. V primeru večjezičnosti je za isto sporočilo vrednost c_msg_id enaka pri vseh jezikih, medtem ko se c_msg spreminja od prevoda do prevoda. Preddefiniran tekst lahko spreminjamo tudi z dodajanjem parametrov (%1, %2, %3, %4, %5), ki se zamenjajo v trenutku vpisa v tabelo plt_log.

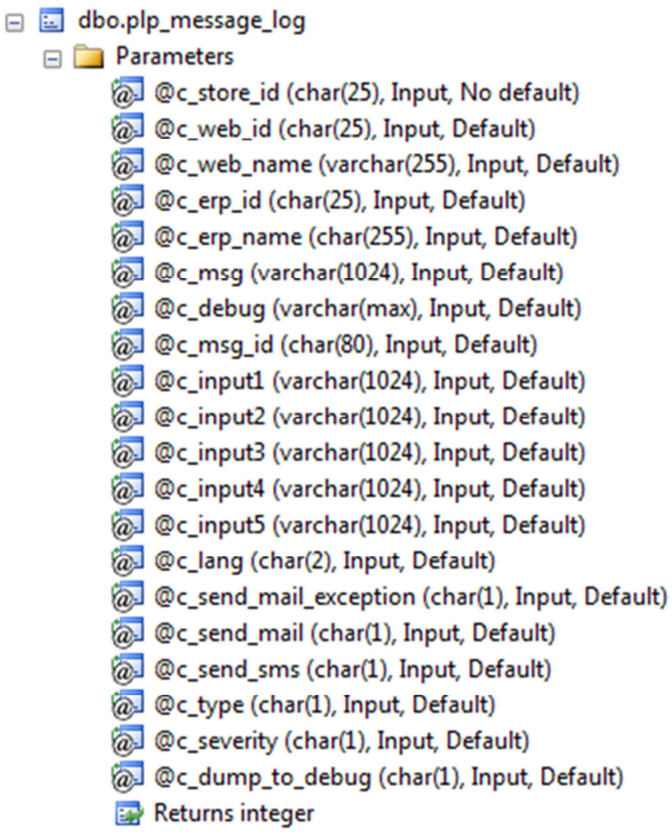
	d_time_ins	c_user_ins	c_msg_id	c_lang	c_msg
235	2015-06-07 18:48:19.463	PLDEV\tomaz	OrderAddItem	EU	Adding new item %1 to set
236	2015-06-07 18:48:19.463	PLDEV\tomaz	OrderAddItem	SI	V šifrant identov, dodajam neobstoječ ident %1
237	2015-06-07 18:48:19.460	PLDEV\tomaz	OrderAddItemError	EU	Error adding an order item. Item is not defined. Web item: %1

Slika 5.16: Primer predlog za večjezikovno vodenje dnevnikov.

plt_message_settings – nastavitve posameznega sporočila. Določimo lahko stopnjo sporočila (0 – proces uspešno zaključen, 1 – obvestilo, 2 – opozorilo, 3 – napaka), kar vpliva na vizualni prikaz v dnevniku dogodkov, tip sporočila oz. sklop sinhronizacije (splošno, naročilo, cene, zaloge ipd.) in na to, ali se za posamezno sporočilo pošlje elektronska pošta oz. SMS-obvestilo ter ali je nastavitev uporabniško spremenjena (se avtomatično nastavi na T, ko uporabnik spremeni parametre sporočila). V primeru uporabniške spremembe se nastavitve ob nadgradnji ne spreminjajo.

plt_log – primarna tabela za vodenje dnevnika dogodkov in obveščanje uporabnika o stanju sinhronizacije.

V dnevnik dogodkov se zapisuje s pomočjo procedure plp_message_log in/ali funkcije plf_message_get. Vhodni parametri procedure so prikazani na Sliki 5.17.



Slika 5.17: Procedura za vpisovanje v dnevnik dogodkov.

Skoraj vsi parametri imajo privzete vrednosti, tako da je uporaba procedure močno poenostavljena. Če npr. želimo sporočiti uporabniku, da se je uspešno sinhronizirala zaloga za artikel s kodo 5 in nazivom »Item Name« v izbranem jeziku, to naredimo na način, kot je prikazano na Sliki 5.18.

```
exec dbo.plp_message_log
    @c_store_id = 'Trgovina'
    ,@c_msg_id = 'StockOK'
    ,@c_input1 = '5'
    ,@c_input2 = 'Item Name'
```

Slika 5.18: Primer klica procedura plp_message_log.

Spremenljivka »StockOK« in vhodna parametra »5« ter »Item name« se bodo v primeru jezika SI prevedli v »Zaloga za ident 5 – Item name uspešno sinhronizirana v spletno trgovino«, v primeru jezika EU pa v »Stock for item 5 – Item name successfully synchronized

to web store«. Na tak način lahko zagotovimo, da se enak klic procedure zapiše v dnevnik dogodkov glede na jezikovne nastavitve.

5.12 Razhroščevanje in obveščanje o napakah

Problem pri razvoju programske opreme oz. njeni aplikaciji med različne uporabnike je vedno nepredvidljivost okolij, v katere se programska oprema namešča. Če želimo ponuditi kakovosten program, ki bo brez večjih problemov deloval pri večini uporabnikov, je nujno treba uvesti poročanje o napakah (crash reporting), tako da avtomatično dobimo povratne informacije o nepredvidenih nepravilnostih, ki se dogajajo pri končnih uporabnikih. Zaradi tega se je v sinhronizator vključila logika spremljanja vseh internih procesov, tako da lahko v primeru napake hitro ugotovimo pot, ki je privedla do napačnega delovanja.

Predvidene napake so seveda del procesa sinhronizacije, katerih logika mora biti nujno vgrajena v program, po drugi strani pa nas vedno zanimajo nepredvidene napake, ki se avtomatično pošiljajo na vnaprej predviden elektronski naslov.

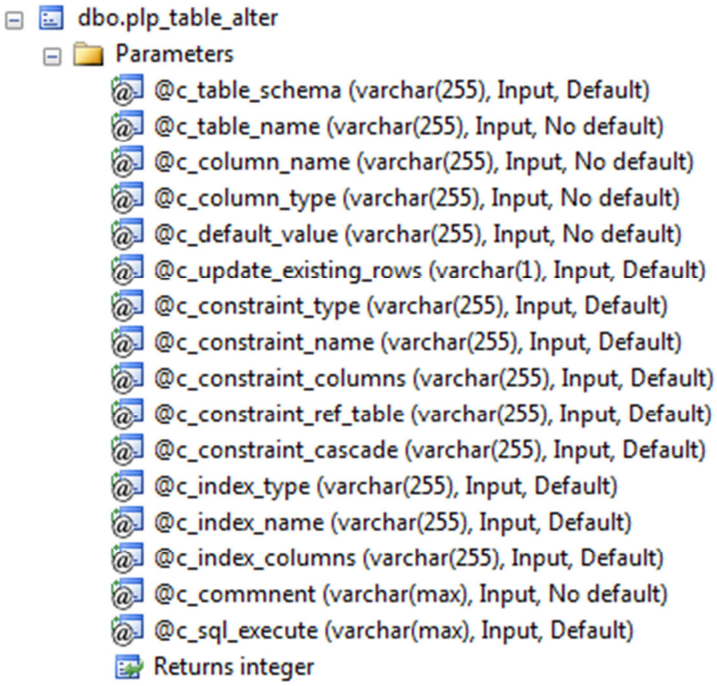
6 Vzdrževanje

6.1 Namestitev in nadgradnje

Avtomatizirana namestitev in nadgradnje so ključnega pomena, če želimo zmanjšati stroške implementacije in s tem omogočiti širšemu krogu uporabnikov, da preveri produkt brez naše pomoči. Na žalost nam pretekle izkušnje kažejo, da je lahko avtomatizirana namestitev sinhronizatorja v tako veliko število različnih informacijskih sistemov s še več različnimi nastavitvami težja, kot se mogoče zdi na prvi pogled. Celotna logika sinhronizacije, ki mora delovati identično na vseh podprtih informacijskih sistemih, se aplicira v procesu namestitve. Tudi najmanjša odstopanja od predvidenega stanja se lahko odražajo v nepravilnem delovanju sinhronizacije. V tem trenutku, četudi bi radi, še ne moremo reči, da je namestitveni čarovnik zaključen, saj bo treba izvesti kar nekaj produkcijskih namestitev, da bomo ugotovili vse zaenkrat še neznane probleme procesa namestitve. Postopek namestitve in nadgradnje je v grobem sestavljen iz spodnjih korakov.

A. Namestitev struktur SQL na strani PIS

Skripte za kreiranje struktur SQL so pisane tako, da omogočajo avtomatično nadgrajevanje, urejanje in dodajanje novih struktur SQL. Definicija tabel se spreminja prek procedure plp_table_alter, ki interno poskrbi, da se dodajo vsi stolpci v tabelo v primeru, da ne obstajajo. Prav tako se lahko dodaja indekse, ključe in dokumentacijo. Vhodni parametri procedure so prikazani na Sliki 6.1.



Slika 6.1: Procedura za kreiranje tabel SQL.

```
exec dbo.plp_table_alter
    @c_table_name = 'plt_settings'
    , @c_column_name = 'c_lang'
    , @c_column_type = 'char(2)'
    , @c_default_value = 'SI'
    , @c_update_existing_rows = 'T'
    , @c_commnent = 'Localization used for log files and user interface. Currently available: SI, EU'
```

Slika 6.2: Preprost primer kreiranja stolpca v tabeli.

Slika 6.3 prikazuje primer, ko želimo dodati index in/ali primarne/sekundarne ključe.

```
exec dbo.plp_table_alter
    @c_table_name = 'plt_stock_external'
    , @c_column_name = 'c_product_id'
    , @c_column_type = 'char(55) not null'
    , @c_default_value = null
    , @c_update_existing_rows = 'F'
    , @c_commnent = 'Item id - it does not need to exist in ERP'
    , @c_constraint_type = 'primary'
    , @c_constraint_name = 'pk_plt_stock_external'
    , @c_constraint_columns = 'c_warehouse_id, c_product_id'
    , @c_constraint_ref_table = ''
    , @c_index_type = 'index'
    , @c_index_name = 'ix_plt_stock_external_product_product_id'
    , @c_index_columns = 'c_product_id'
```

Slika 6.3: Kompleksen primer kreiranja stolpca v tabeli.

Ker je logika narejena prek procedure, bi vsaj v teoriji moral biti prehod na drug strežnik SQL lažje izvedljiv, vendar bo za podrobno izkušnjo treba počakati na praktični primer (zaenkrat je na strani PIS podprt samo Microsofov strežnik SQL).

B. Prenos datotek

Vse izvedljive datoteke (EXE) so samostojne celote, ki se jih nadgradi tako, da staro verzijo prepišemo z novo. Seveda moramo paziti, da so strukture SQL v bazi ustrezno usklajene z izvedljivimi datotekami, kar je naloga namestitvenega čarovnika.

C. Namestitev na strani ST

Namestitev na strani ST pomeni kreiranje struktur SQL in kopiranje skript (PHP) v svoj direktorij znotraj trgovine ter nastavljanje opravilnika cron za zagon avtomatičnega izvajanja, ker je možnih načinov prenosa na spletni strežnik veliko (FTP, SSH, SFTP, SMB). Posledično za veliko potencialnih problemov trenutno ni v planu, da bi se ta korak avtomatiziral. Sinhronizator bo v procesu namestitve na strani PIS odložil datoteke v svoj direktorij, ki ga bo uporabnik ročno prenesel v trgovino.

Kot zanimivost lahko omenimo, da smo dolgo časa iskali ustrezno programsko orodje, ki bi nam omogočalo vizualno urejanje struktur SQL (diagrame ER s pomočjo orodij UML). Preverili smo najbolj znane ponudnike (Sybase PowerDesigner, VisualParadim, Sparx Enterprise Architect ipd.) in nobeden ni omogočal kreiranja skript SQL iz diagramov ER, ki bi hkrati omogočale tudi preproste nadgradnje iz katere koli verzije, tako kot smo kasneje realizirali sami (s pomočjo procedure plp_table_alter). Vsi so sicer ponudili možnost generiranja skript SQL za kateri koli strežnik SQL, vendar je bila vedno domneva, da je baza prazna in da se vsi objekti kreirajo na novo oz. da se posodablja bazo, na katero je orodje UML povezano. To pa na žalost pomeni vpeljavo zelo stroge evidence verzij struktur SQL (vsaka nova sprememba na bazi bi pomenila novo verzijo), kar pa kliče po napakah in nepotrebnih »birokratskih« opravilih.

6.1.1 Uporabniške nadgradnje

Postopek namestitve predvideva tudi spremembe s strani uporabnika oz. mu daje možnost spremeniti način delovanja sinhronizatorja. Spremeni lahko logiko skoraj vseh struktur SQL (procedure, funkcije, pogledi, pogojno tabele). Če so spremembe pravilno označene, bodo ostale tudi po nadgradnji. V procesu nadgradnje se namreč na novo kreira vse procedure, funkcije in poglede ter doda nove stolpce v tabele. Primer, kako se kreira procedura, je prikazan na Sliki 6.4.

```
-- CREATE PROCEDURE IF NOT EXISTS
if not exists (select name from sysobjects where name = 'plp_prepare_stock' and type = 'P')
exec('create procedure dbo.plp_prepare_stock as')
GO
-- ALTER PROCEDURE
alter procedure dbo.plp_prepare_stock
with encryption
AS
```

Slika 6.4: Primer kreiranja objekta SQL.

V prvem koraku kreiramo prazno proceduro, če ne obstaja. V drugem koraku prepišemo proceduro na zadnjo verzijo in tako prepišemo kakršne koli spremembe, ki/če so bile delane na proceduri.

Če želi uporabnik spremeniti delovanje katere koli procedure/funkcije/pogleda in ohraniti spremembe po nadgradnji, je treba narediti sledeče:

- spremeni originalno proceduro, s tem da mora paziti, da so vhodno/izhodni parametri kompatibilni s sinhronizatorjem. Večinoma to pomeni enake vhodno/izhodne parametre, razen v primeru pogledov, kjer lahko stolpce poljubno dodajamo, vendar morajo biti imenovani s predpono »u_«;

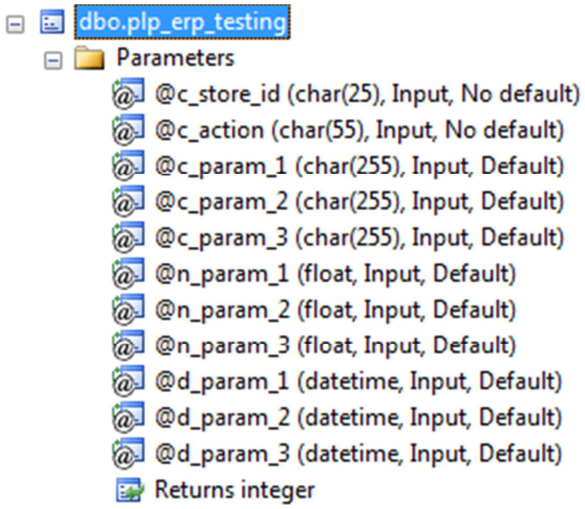
- ko naredi spremembe objekta SQL, mora kreirati istoimensko kopijo z dodano pripono »_user«. V primeru na Sliki 6.4 to pomeni, da bi v bazi morala obstajati procedura plp_prepare_stock_user. V procesu namestitve se bo namreč na koncu izvedel prepis vseh sistemskih procedur/funkcij/pogledov z uporabniškimi.

6.2 Testiranje

Po vsaki novi namestitvi je treba preveriti, ali sinhronizacija deluje po pričakovanjih, in odpraviti morebitne napake, ki se pojavijo zaradi specifičnosti uporabniške baze oz. nepravilne namestitve komponent. Zaradi kompleksnosti in heterogenosti celotnega sistema bi bilo zelo težko sprogramirati orodje, ki bi služilo za testiranje vseh funkcionalnosti od njihovega izvora (zaslonska maska PIS) do ponora (prikaz podatkov končnemu kupcu v ST), in to predvsem zato, ker sinhronizator predstavlja »samo« vmesni člen ali vezivo med PIS in ST, medtem ko na oba ključna/končna sistema ni direktnega vpliva. Prav tako obstaja ideja, da bodo integracijo na strani PIS vzdrževali njihovi razvijalci (Datalab, Saop, Vasco ipd.) glede na podano dokumentacijo strani avtorja sinhronizatorja. Zato je edino smiselno, da se vzpostavi testno okolje, ki preverja pravilnost procesov na nivoju baze SQL oz. na nivoju, za

katerega je »odgovoren« sinhronizator, medtem ko se aplikacija podatkov v ST in PIS preverja vizualno (ročno).

Za namene testiranja se na strani PIS kreira procedura, ki sledi logiki plt_sync_queue, in sicer kot parameter prejema kodo trgovine, tip akcije in določene kriterije.



Slika 6.5: Procedura za testiranje funkcionalnosti.

Testiranje se izvaja tako, da se na ravni baze simulira (uporabniško) spremembo podatkov in preverja, ali se izvedejo vsi potrebni procesi, ki bi tej spremembi morali slediti. Tako lahko npr. sprožimo preverjanje zaloge za en ident, kot je prikazano na Sliki 6.6.

```
exec plp_erp_testing
    @c_store_id = null
    ,@c_action = 'Stock_toWS'
    ,@c_param_2 = 'X1/15' -- c_product_sku
```

Slika 6.6: Primer klica procedura za testiranje.

V primeru Pantheona procedura interno izvede spremembo zaloge v »matični« tabeli za en artikel +1/–1, kar bi moralo biti dovolj, da se sproži ponovna sinhronizacija za ta artikel.

```
if @c_action = 'Stock_toWS' begin
    set @nTop = convert(int,isnull(@n_param_1,1))
    if @nTop = -1
        set @nTop = 999999999
    -- SELECT ITEM(S)
    insert into #t_items(acIdent, acWareHouse)
    select top (@nTop) ST.acIdent, ST.acWareHouse
        from tHE_Stock ST
        join tHE_SetItem SI on SI.acIdent = ST.acIdent
        join plt_item_settings SE on SE.c_product_id = SI.anQid
            where (SE.c_sync_stock = 'T')
            and ((@c_param_1 = '') or (SI.anQId = @c_param_1))
            and ((@c_param_2 = '') or (SI.acIdent = @c_param_2))

    -- SIMULATE ERP CHANGE (change qty +1/-1 forcing trigger to be executed or last stock change time should be updated)
    update tHE_Stock set anStock = anStock + 1
        from tHE_Stock ST
        join #t_items I on I.acIdent = ST.acIdent and I.acWareHouse = ST.acWareHouse

    update tHE_Stock set anStock = anStock - 1
        from tHE_Stock ST
        join #t_items I on I.acIdent = ST.acIdent and I.acWareHouse = ST.acWareHouse

    -- LOG - FOR DEBUGGING PURPOSES
    set @c_tmp = 'Updating stock (+1/-1 qty) for '+convert(varchar,@@rowcount)+' item(s)'
    exec plp_log_sql @c_tmp
end
```

Slika 6.7: Prikaz dela kode procedure za testiranje.

6.3 Dokumentacija

Produkt se primarno razvija na Microsoftovem strežniku SQL, ki omogoča tudi dokumentiranje vseh struktur SQL, kar smo izkoristili za namene razvojne dokumentacije. Kot je že prikazano na prejšnjih primerih, se za kreiranje struktur SQL uporablja standardizirana procedura, ki med drugimi kot parameter pričakuje tudi spremenljivko c_comment. V to spremenljivko lahko vpišemo opis objekta (procedura, pogled, tabela, stolpec ipd.), procedura pa interno poskrbi, da se komentar doda na nivoju baze SQL v definicijo objekta. Tako lahko s klicem ene skripte pridobimo vse informacije o vseh naših dokumentiranih strukturah SQL, kar je seveda zelo koristno pri vzdrževanju dokumentacije. Predvsem pa je to zanimiv pristop, saj se dokumentacija piše sproti skupaj s programsko kodo in ni ločen proces, na katerega bi sicer lahko pozabili.

Primer dokumentacije za dve tabeli SQL, ki je na voljo »na klik« v obliki TXT, je prikazan na Sliki 6.8.

```
plt_lang (user_table)Language options
        d_time_ins - Datetime when row record was added to table
        c_user_ins - SQL user who inserted the record
        c_lang - Localization used for log files and user interface. Currently available: SI, EU
        c_name -

plt_log (user_table)Table for logging events
        n_id - Autoincrement id
        d_time_ins - Datetime when row record was added to table
        c_user_ins - SQL user who inserted the record
        c_store_id - Store id in synchronizator (plt_settings_store.c_store_id). Licence key is
                     linked to c_store_id, so IT SHOULD NOT BE CHANGED once licence keys are
                     generated
        c_type - Type of log entry (orders, stock, items, general infor, web store info, ...)
        c_web_id - Web id of synchronization entity
        c_web_name - Web name of synchronization entity
        c_erp_id - ERP id of synchronization entity
        c_erp_name - ERP name of synchronization entity
        c_severity - Severity of log entry: 1 - general info, 2 - warning, 3 - error, 0 - Successful
                     completition of operation/batch (order import, stocku update)
        c_msg - Message of the log entry. Localization of log messages can be used with the help of
                procedure
        c_debug - Debug info - it should be filled when severity = 3
        c_send_mail - Wheather to send mail message to the receiver defined in
                      plt_settings.c_mail_address. When mail is sent, value is changed from T to F
        c_send_mail_time - Marks when mail was sent.
        c_send_mail_exception - Program exceptions are automatically sent to bug system for futher
                                processing. When mail is sent, value is changed from T to F
        c_send_mail_exception_time - Marks when mail was sent.
        c_send_sms - Wheather to send sms message to the receiver defined in
                     plt_settings.c_sms_number. When sms is sent, value is changed from T to F
        c_send_sms_time - Marks when sms was sent.
```

Slika 6.8: Dokumentacija struktur SQL.

7 Sklepne ugotovitve

Kot vsaka programska oprema ima tudi sinhronizator ogromno možnosti za izboljšave in spremembe, še posebej ker je njegova zasnova zastavljena izredno široko in predstavlja praktično prvo verzijo v novi funkcionalni strategiji razvoja. V diplomskem delu predstavljen sinhronizator je že četrta večja funkcionalna različica. Prvoten razvoj se je začel že konec leta 2009, ko se je razvila prva verzija, predvsem za lastne potrebe. Kasneje se je razvoj razširil na bolj generično rešitev. Naredil se je prehod z »linked« serverjev na bolj robustne komponente (Delphi). Vpeljale so se vmesne strukture SQL za lažjo horizontalno razširljivost, in to vse do trenutne točke, ko je sinhronizator dejansko postal lastna entiteta v sinhronizacijskem procesu, neodvisna tako od PIS kot od ST, in ponuja rešitev »out of the box« za sinhronizacijo.

Do sedaj sinhronizator v taki ali drugačni obliki uporablja približno 40 do 50 slovenskih in tujih podjetij. Na naše veselje se zelo redko zgodi, da programska oprema ne bi delovala tako, kot je bila zamišljena (uporabniki skoraj nikoli ne pokličejo zaradi nedelovanja programske opreme). Največkrat se zgodi, da je treba sinhronizator nadgraditi zaradi nepredvidene nastavitve v ST, ki funkcionalno »ruši« privzet način delovanja. Občasno so problemi v procesu namestitve, ki nastanejo kot posledica neustreznih nastavitev spletnega strežnika oz. preobsežnih jedrnih (core) sprememb ST, vendar so to zaradi same zasnove lahko rešljive napake, ko enkrat poznamo njihov vzrok.

V katero smer se bo rešitev razvijala naprej, bodo določale predvsem tržne okoliščine, vsekakor pa bo poleg spremenjenega funkcionalnega pristopa primaren fokus širjenje med različne PIS, podpora čim večjemu številu ST, njegova internacionalizacija in čim bolj obširna dokumentacija.

Literatura

[1] R. Dewson, *Beginning SQL Server 2005 For Developers*, New York: Springer-Verlag, 2006.

[2] CodeGear, RAD Studio VCL Reference, 2008

[3] A. Stage. Synchronization and replication in the context of mobile applications (2005). *Technische Universität München* [Online]. Dosegljivo: http://www14.in.tum.de/konferenzen/Jass05/courses/6/Papers/11.pdf.

[4] A. McCombs in R. Banh, *The Definitive Guide to Magento: A Comprehensive Look at Magento*, New York: Springer-Verlag, 2009.

[5] R. Gudakesa, M. Sukarsa, Gusti Made Arya Sasmitaa, »Two-ways database synchronization in homogeneous dbms using audit log approach«, *Journal of Theoretical and Applied Information Technology* [Online], št. 3, zv. 65, str. 854-859, 2005. Dosegljivo: http://www.jatit.org/volumes/Vol65No3/31Vol65No3.pdf.

[6] SQL Server Books Online. *Microsoft SQL Server* [Online]. Dosegljivo: https://technet.microsoft.com/en-us/library/ms130214%28v=sql.105%29.aspx.

[7] V. Šinigoj in M. Mertelj. PANTHEON 5.5 Uporabniški priročnik. (2010). *Datalab* [Online]. Dosegljivo: FTP://FTP.datalab.si/Marketing/Uporabniski_prirocnik/PANTHEON-prirocnik-za-zacetnike.pdf.

[8] M. Wiesmann, F. Pedone in A. Schiper. Understanding Replicationin Databases and Distributed Systems. *Swiss Federal Institute of Technology* [Online]. Dosegljivo: http://www-users.cselabs.umn.edu/classes/Spring-2014/csci8980-sds/Papers/ProcessReplication/Understanding-Replication-icdcs2000.pdf.

[9] Universal Data Access Components. *Devart* [Online]. Dosegljivo: https://www.devart.com/unidac.

[10] Online Help for Delphi® XE7 and C++Builder® XE7. *Embarcadero* [Online]. Dosegljivo: http://docwiki.embarcadero.com/RADStudio/XE7/en/Main_Page.

[11] Data synchronization. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Data_synchronization.

[12] Synchronization (computer science). *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Synchronization_%28computer_science%29.

[13] Open Database Connectivity. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Open_Database_Connectivity.

[14] What is SymmetricDS. *SymmetricdsDS* [Online]. Dosegljivo: http://www.symmetricds.org.

[15] Who Is Talend? *Talend* [Online]. Dosegljivo: https://www.talend.com.

[16] Shadowbase Solutions - Data Replication and Data Integration Software. *Gravic, Inc*. [Online]. Dosegljivo: http://www.gravic.com/shadowbase/solutions/overview.html.

[17] Blowfish (cipher). *Wikipedia* [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Blowfish_%28cipher%29.

[18] Use SQL Server Management Studio. *Microsoft: Developer Network* [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/ms174173.aspx.

[19] Microsoft SQL server. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Microsoft_SQL_Server.

[20] MySQL. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/MySQL.

[21] The Best MySQL GUI Tool You Can Find. *Devart* [Online]. Dosegljivo: https://www.devart.com/dbforge/mysql/studio.

[22] Delphi (programming language). *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Delphi_%28programming_language%29.

[23] Pantheon Enterprise. *Datalab* [Online]. Dosegljivo: http://www.datalab.si/pantheon/enterprise.

[24] Inno Setup. *Jordan Russell's software* [Online]. Dosegljivo: http://www.jrsoftware.org/isinfo.php.

[25] E-commerce Platforms Popularity Study, October 2014. *AheadWorks magento extensions* [Online]. Dosegljivo: http://blog.aheadworks.com/2014/10/e-commerce-platforms-share-investigation-october-2014.

[26] Vodič po Datalab PANTHEON-u 5.5. *Pantheon u site* [Online]. https://usersite.datalab.eu/Wiki/tabid/178/language/sl-SI/Default.aspx.

[27] Java Database Connectivity. *Wikipedia* [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Java_Database_Connectivity.

[28] FireDAC Multi-Device Data Access Library. *Embarcadero* [Online]. Dosegljivo: http://www.embarcadero.com/products/rad-studio/firedac.

[29] Zakon o varstvu potrošnikov. *Uradni list RS* [Online]. Dosegljivo: https://www.uradni-list.si/1/content?id=51154.

[30] Replication versus Synchronization. *Pervasync Blog* [Online]. Dosegljivo: https://pervasync.wordpress.com/2011/08/17/server-centric-pervasive-computing-and-data-synchronization.

[31] Cloud ERP for growing your business. *ERPin Cloud* [Online]. Dosegljivo: http://www.erpincloud.com.

[32] Microsoft .NET. *Wikipedija* [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Microsoft_.NET.

[33] Programski jezik Java. *Wikipedija* [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Programski_jezik_java.

[34] PHP. *Wikipedija* [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/PHP.